Science

Michael Langbein

February 19, 2019

Contents

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 *imageine 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:* $\forall x, y, z \in S : xRy \land yRz \rightarrow xRz$

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:* $\forall a, b \in S : aRb \lor bRa$

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 comvolutions

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

The combination , aka. the *binomial coefficient*, for drawing k items out of n items, is defined as:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Permutations are the number of rearrangements. Permutations of unique items are relatively straightforeward, but things get a little involved when we deal with permutations of grouped items.

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 inducion on the number of groups i. In the base-case with two groups we trivially get the binomial coefficient. Then do the induction step.

1.3.5 Generating functions

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 a more 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 to buy 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. □

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 $B : baseV \leftrightarrow (B : li \wedge B : spanV)$.

The dimension of a vecotrspace S is defined as the size of its base B. 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 $\vec{a} : ld(\vec{b}, \vec{c}) \vee \vec{b} : ld(\vec{a}, \vec{c}) \vee \vec{c} : ld(\vec{a}, \vec{b})$

□

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 8 (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.
Proof that $B : orth \rightarrow B : li$

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

Thus $B : orth \rightarrow B : li$

□

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 take a step back and only consider vector spaces, not requireing them to also be equipped with an inner product.

1.5.1 Matrix properties

Matrices form a vector space This can be easily proven.

The set of all $n \times m$ matrices M is a vector space. For this, we just need to go through the seven properties from the previous chapter.
Proof that scalar multiplication properties hold

> Proof that M is closed under scalar multiplication
> |
> Thus M is closed under scalar multiplication
> Proof that $a(b\mathbf{m}) = (ab)\mathbf{m}$
> |
> Thus $a(b\mathbf{m}) = (ab)\mathbf{m}$

Thus scalar multiplication properties hold

Proof that vector addition properties hold
|
Thus vector addition properties hold

Proof that vector addition and scalar multiplication are distributive
|
Thus vector addition and scalar multiplication are distributive

□

Properties of matrices will be proven here.
Here is a little exercise to get you warmed up.

If $\mathbf{A}\vec{x}$ is zero for any $\vec{x}$, then $\mathbf{A}$ must be the zero matrix. Proof that $\forall \vec{x} : \mathbf{A}\vec{x} = \vec{0} \to \mathbf{A} = \mathbf{0}$

> Let $\vec{x}_0$ be chosen. Suppose $\mathbf{A}\vec{x}_0 = \vec{0}$. Proof that $\mathbf{A} = \mathbf{0}$
> |
> Thus $\mathbf{A} = \mathbf{0}$

Thus $\forall \vec{x} : \mathbf{A}\vec{x} = \vec{0} \to \mathbf{A} = \mathbf{0}$

□

Here is a method to prove that two matrices are identical.

Proof. Proof that $\forall \vec{x} : \mathbf{A}\vec{x} = \mathbf{B}\vec{x} \to \mathbf{A} = \mathbf{B}$

> Let $\vec{x}$ be chosen. Suppose $\mathbf{A}\vec{x} = \mathbf{B}\vec{x}$ Proof that $\mathbf{A} = \mathbf{B}$
> |
> Thus $\mathbf{A} = \mathbf{B}$

Thus $\forall \vec{x} : \mathbf{A}\vec{x} = \mathbf{B}\vec{x} \to \mathbf{A} = \mathbf{B}$

□

Matrix inverse and transpose will be handled here.

1.5.2 Important *non*properties

1.5.3 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 $\mathbf{B}^{-1}\mathbf{A} = \mathbf{T}_{BA}$:

$$\begin{aligned}
\mathbf{B}^{-1}\mathbf{A}(\vec{v})_A &= \mathbf{T}_{BA}(\vec{v})_A \\
&= \sum_n (v_n)_A (\vec{a}_n)_B \\
&= \sum_n (v_n)_A \mathbf{B}^{-1}\mathbf{A}(\vec{a}_n)_A \\
&= \mathbf{B}^{-1}\mathbf{A} \sum_n (v_n)_A (\vec{a}_n)_A \\
&= \mathbf{B}^{-1}\mathbf{A}\ \mathbf{I}(\vec{v})_A \\
(\vec{v})_A &= (\vec{v})_A
\end{aligned} \tag{1.1}$$

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

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

1.5.4 Linear transformations

Definition 9. *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.

1.5.5 Systems of linear equations

1.6 Geometric algebra

The geometric algebra is the algebra of multivectors. multivectors form a vectorspace, an inner product space and an algebra.

1.6.1 Definition of the algebra

The geometric algeba has elements

$$A = scalar + vector + plane + volume + ... = (A)_0 + (A)_1 + (A)_2 + (A)_3 + ...$$

It can be proven that such an algebra exists with the following gemoetric product:

- $AB \in \mathbb{G}^n$
- $A(B + C) = AB + BC$
- $(A + B)C = AC + BC$
- $(\alpha A)B = A(\alpha B) = \alpha(AB)$
- $(AB)C = A(BC)$
- $1A = A1 = A$

1.6.2 Canonical basis

In $\mathbb{G}^2$, this basis would consist of $1, e_1, e_2, e_1e_2$.
A little slang is in order:

- a k-vector is a multivector consisting only of rank-j elemens. For example, a 2 vector in $\mathbb{G}^3$ could be written as $a = \alpha e_1e_2 + \beta e_2e_3 + \gamma e_3e_1$

1.6.3 Geometric product

Let's inspect the case of $\mathbb{G}^1$.

$$\begin{aligned} A &= \alpha_0 + \alpha_1 e \\ B &= \beta_0 + \beta_1 e \\ AB &= (\alpha_0 + \alpha_1 e)(\beta_0 + \beta_1 e) \\ &= (\alpha_0\beta_1 + \alpha_1\beta_0) + (\alpha_1\beta_0 + \alpha_0\beta_1)e \end{aligned} \tag{1.2}$$

1.6.3.1 Inner product

The inner product of a j-vector A and a k-vector B is:

$$A \cdot B = (AB)_{\text{rank}(B)-\text{rank}(A)}$$

1.6.3.2 Outer product

The outer product of a j-vector A and a k-vector B is:

$$A \wedge B = (AB)_{\text{rank}(A)+\text{rank}(B)}$$

1.6.3.3 Norm and inverse

The reverse of a k-vector $A = a_1 a_2 ... a_k$ is $A^\dagger = a_k ... a_2 a_1$. This equals $A^\dagger = (-1)^{k(k-1)/2} A$.

The norm of a multivector is defined as

$$A = \sum_J \alpha_J e_J \to |A| = \sum_J \alpha_J^2$$

If A is a k-vector $A = a_1 a_2 ... a_k$, then the inverse of A is

$$A^{-1} = \frac{(-1)^{k(k-1)/2}}{|A|^2} A$$

If B isn't a k-vector, then it can be expressed as as sum of k-vectors. But be careful: the norm is not a linear function. So $|B| = |\sum_j A_j| \neq \sum_j |A_j|$

1.6.3.4 Parallel and orthogonal part

1.6.4 Fourier decomposition of multivectors

Every multivector can be written in the form

$$A = \sum_J (e_J^{-1} A)_0 e_J$$

This is easily proven.

$$\begin{aligned} A &= \sum_J \alpha_J e_J \\ (A e_I)_0 &= \sum_J \alpha_J (e_J e_I)_0 \\ &= \alpha_I \end{aligned} \tag{1.3}$$

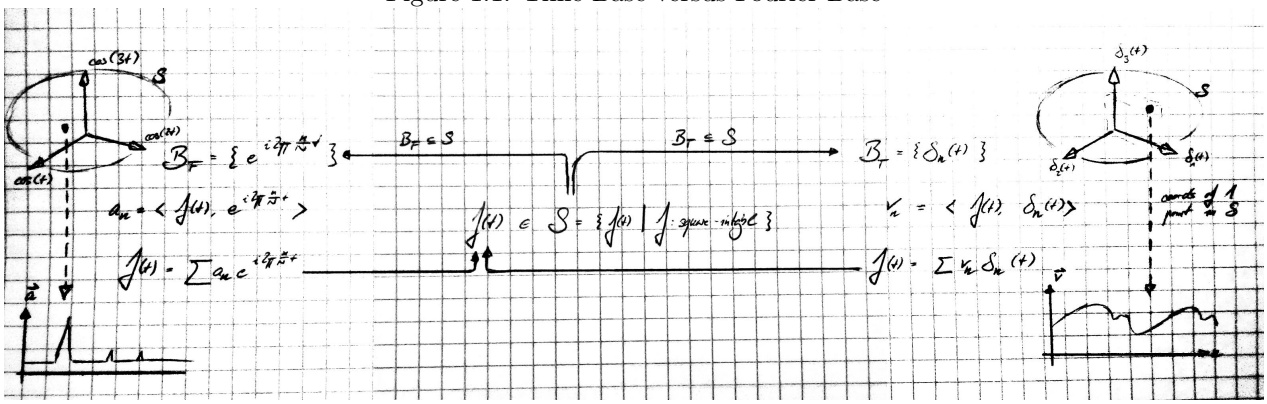
If a is a vector, this reduces to the normal

$$a = \sum_j (a \cdot e_j) e_j$$

1.6.5 Geometric calculus

1.7 Fourier and Laplace

Figure 1.1: Time Base versus Fourier Base



1.7.1 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 samplesize and δ be the samplefrequency (often equal to 44100 Hz or 48000 Hz). Let V be the samplespace of signals of size N. This is a vectorspace, 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.

1.7.1.1 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 signalvector and n will be the index of frequency/the basevectors.

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

1.7.1.2 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 Horners method, this is a $\Theta(n)$ operation. Executing it on all n coefficients, the whole proces 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.

1.7.2 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:

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

□

1.7.3 Fast Fourier Transform

In the previous section we have evaluated the polynomal $\sum_{k=0}^{N-1} v_k e^{i2\pi \frac{n}{N} k}$ 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 polynomal $A(x) = \sum_{k=0}^{N-1} v_k x^k$, 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 polynomal $A(x)$, 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:

$$\begin{aligned} A(w_N^n) &= A_{even}(w_N^{2n}) + w_N^n A_{odd}(w_N^{2n}) \\ &= A_{even}(w_{N/2}^n) + w_N^n A_{odd}(w_{N/2}^n) \end{aligned} \tag{1.4}$$

$$\begin{aligned} A(w_N^{n+N/2}) &= A_{even}(w_N^{2n+N}) + w_N^{n+N/2} A_{odd}(w_N^{2n+N}) \\ &= A_{even}(w_N^{2n}) - w_N^n A_{odd}(w_N^{2n}) \\ &= A_{even}(w_{N/2}^n) - w_N^n A_{odd}(w_{N/2}^n) \end{aligned} \tag{1.5}$$

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
```

1.7.3.1 Backtransformation

1.7.4 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.

1.7.4.1 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.

1.7.4.2 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.

$$\begin{aligned} \vec{s} &= \sum_F a_f e^{-2\pi i f t} \text{ , with F = \{440, 465, ...\}} \\ &= \begin{bmatrix} ... & & ... \\ & e^{-2\pi i f t} & \\ ... & & ... \end{bmatrix} \vec{a} \\ \vec{a} &= \begin{bmatrix} ... & & ... \\ & e^{-2\pi i f t} & \\ ... & & ... \end{bmatrix}^{-1} \vec{s} \end{aligned} \tag{1.6}$$

1.7.5 Continuous Fourier Analysis

In Fourier analysis, we deal with the innerproduct space $C^2_{[a,b]}$: the space of square-integrable functions that are continuous from a to b. An orthogonal base for this vectorspace 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 Eulers formula, we instead write this base as $\{e^{i\frac{n2\pi}{b-a}t} | j \in \mathbb{N}\}$.

1.7.5.1 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 1.4: Fourier versus Taylor

	coefficients	basevectors
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 datamatrix. It turns out that we can use the set of eigenvectors of the datamatrix[2].

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

□

1.7.5.2 Proving that Fourier functions form a basis

We can use Fourier to describe any functionspace - 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.

[1] proof required

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

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/

1.7.5.3 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.

$$\begin{aligned} g : A^N &\to A^M \\ g(\vec{x}) &= \sum_m^M g_m(\vec{x})\vec{e}_m \text{ ... decomposing the function over the output basis} \\ \mathscr{F}_g(\vec{u}) &= \sum_m^M \vec{e}_m \mathscr{F}_{g_m}(\vec{u}) \text{ ... Fourier transform over the input basis} \end{aligned} \tag{1.7}$$

Consider the example of a rgb-image.

$$\begin{aligned} c(x,y) &= \vec{e}_1 r(x,y) + \vec{e}_2 g(x,y) + \vec{e}_3 b(x,y) \\ \mathscr{F}_c \begin{bmatrix} f_x \\ f_y \end{bmatrix} &= \int_X \int_Y c(x,y) e^{-2\pi i (f_x x + f_y y)} \, dy \, dx \\ &= \int_X \int_Y \vec{e}_1 r(x,y) e^{-2\pi i (f_x x + f_y y)} \, dy \, dx + ... \\ &= \vec{e}_1 \mathscr{F}_r \begin{bmatrix} f_x \\ f_y \end{bmatrix} + \vec{e}_2 \mathscr{F}_g \begin{bmatrix} f_x \\ f_y \end{bmatrix} + \vec{e}_3 \mathscr{F}_b \begin{bmatrix} f_x \\ f_y \end{bmatrix} \end{aligned} \tag{1.8}$$

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

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

1.8 Graphics

1.8.1 Points

Distance in 1-D:

$$d = \Delta x$$

Distance between two points in 2-D:

$$d = \sqrt{\Delta x^2 + \Delta y^2}$$

Distance between two points in 3-D:

$$d = \sqrt{\sqrt{\Delta x^2 + \Delta y^2}^2 + \Delta z^2} = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$$

1.8.2 Geometric objects in matrix notation

$x = 1$ defines a plane, as does $x = 2y$. $x = 1 \wedge y = 2$ defines a line. More generally, the dimension of a geometrical object equals 3 minus the number of equations needed to describe the object. Informally written: $dim(obj) = 3 - \#(=)$.

In more rigorous terms, the "number of equations needed to describe the object" is called the rank of the matrix. 3 in our case is the dimension of the space. The geometrical object is really the column-space of the matrix.

1.8.2.1 Plane

Defined by one point $\vec{b}$ and two lines $\vec{p_1}, \vec{p_2}$:

$$P = \{\vec{x} | \exists a, b : \vec{x} = \vec{b} + a\vec{p_1} + b\vec{p_2}\}$$

Or defined by one point $\vec{b}$ and one normal $\vec{n}$:

$$P = \{\vec{x} | (\vec{x} - \vec{b}) \cdot \vec{n} = 0\}$$

1.8.2.2 Line

Defined by two points:

$$L = \{\vec{x} | \exists \alpha : \vec{x} = \vec{b} + \alpha \cdot \vec{\Delta}\}$$

Or transformed in matrix-notation:

$$\vec{x} = \begin{bmatrix} \vec{b} & \vec{\Delta} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ \alpha \end{bmatrix}$$

$$\begin{bmatrix} \vec{b} & \vec{\Delta} \end{bmatrix}^{-1} \cdot \vec{x} = \begin{bmatrix} 1 \\ \alpha \end{bmatrix}$$

1.8.2.3 Intersection Line/Plane

Substituting the definition of a line into the definition of a plane we get:

$$(\vec{b_l} + \alpha \cdot \vec{\Delta} - \vec{b_p}) \cdot \vec{n} = 0$$

This reduces to:

$$\alpha = \frac{(\vec{b_l} - \vec{b_p}) \cdot \vec{n}}{\vec{\Delta} \cdot \vec{n}}$$

Alternatively, we can make use of the two-line-definition of the plane and obtain:

$$\vec{b} + a\vec{p_1} + b\vec{p_2} = \vec{b} + \alpha \cdot \vec{\Delta}$$

$$\vec{b_l} - \vec{b_p} = \begin{bmatrix} \vec{p_1} & \vec{p_2} & \vec{\Delta} \end{bmatrix} \begin{bmatrix} a \\ b \\ \alpha \end{bmatrix}$$

1.8.3 Projections

1.8.3.1 Finding a perpendicular

Having a vector $\vec{a}$, how do we find a vector $\vec{a^T}$ that has the following properties:

- $\vec{a} \cdot \vec{a^T} = 0$
- $|\vec{a}| = |\vec{a^T}|$

These two requirements can be re-expressed as

- $ax + by = 0$
- $x^2 + y^2 = a^2 + b^2$

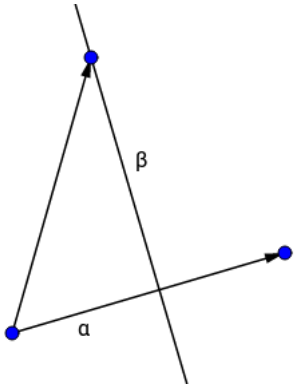
This is a nonlinear system, but still solveable. Indeed, we quickly find:

$$\vec{a^T} = \begin{bmatrix} -y \\ x \end{bmatrix}$$

How about the two perpendiculars to a 3d-vector? You can just use $\begin{bmatrix} -y & x & 0 \end{bmatrix}$ and $\begin{bmatrix} 0 & -z & y \end{bmatrix}$. All possible perpendiculars will be linear combinations of these two.

1.8.3.2 Projecting one vector onto another

Imagine we wanted to know α in the following graphic:



Here, α is the length of vector $\vec{b}$ projected onto $\vec{a}$. How can we find α?

The solution lies in imagening the vector $\vec{b}$ being expressed inside a coordinate-system consisting of $\vec{a}$ and $\vec{a^T}$, like so:

$$[\vec{a}, \vec{a^T}] \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \vec{b}$$

1.8.4 Implicit versus parameterized representation of bodies

In geometry, we often represent bodies in set-notation, aka. implicit notation. Consider for example the ellipsoid:

$$\left\{ \vec{v} \middle| \frac{x^2}{r_1^2} + \frac{z^2}{r_2^2} + \frac{z^2}{r_3^2} = 1 \right\}$$

This body can be parameterized as:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r_1 \cos\theta \cos\phi \\ r_2 \cos\theta \sin\phi \\ r_3 \sin\theta \end{bmatrix}$$

In the set notation, the ellipsoid had no parameters (it had r_1, r_2, r_3, but these are *constants*). With θ and ϕ we now found two parameters which, when varied over their whole domain $[0°, 360°]^2$, yield every point in the set.

Parameterisation means *finding a function of one or more parameters who's range equals the set.* That means nothing other than *putting the vector $\vec{x}$ on the left side.* Conversely, you have an *implicit* equation when you can write the equation such that the left side is 0 (for clarity: we mean the number 0, not a zero-vector).

As a sidenote, when you find a set-expression of a body that contains a *thereis* statement, chances are that this statement already *is* parameterized.

Table 1.5: Explicit and parametric set of plane and ellipsoid

	implicit	parametric
linear	$\left\{\vec{x} \middle\vert (\vec{x}-\vec{b})\vec{\cdot}\vec{n}=0\right\}$	$\left\{\vec{x} \middle\vert \exists\alpha,\beta : \vec{x}=\begin{bmatrix}\vec{b} & \vec{p}_1 & \vec{p}_2\end{bmatrix}\begin{bmatrix}1\\ \alpha\\ \beta\end{bmatrix}\right\}$
non-linear	$\left\{\vec{x} \middle\vert \frac{x^2}{r_x}+\frac{y^2}{r_y}+\frac{z^2}{r_z}=1\right\}$	$\left\{\vec{x} \middle\vert \exists\theta,\phi : \vec{x}=\begin{bmatrix}r_x\cos\theta\cos\phi\\ r_y\cos\theta\sin\phi\\ r_z\sin\theta\end{bmatrix}\right\}$

We can find more about this topic here: https://en.wikipedia.org/wiki/Implicit_surface. In this section, we have been careful to never mention the term *explicit.* In this section, we dealt with bodies. But if we only wanted surfaces, that is, bodies where there are never two z values at on x/y spot, there is also the explicit representation $z = f(x, y)$.

Table 1.6: Overview of implicit, explicit and parameterized functions

	in	out	notes
implicit	x, y, z		
explicit	x, y	z	doesn't allow more than one z per x/y
parameterized	θ, ϕ	x, y, z	

1.9 Probability

1.9.1 Basics

1.9.1.1 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 10 (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 11 (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 12 (Expectation)**.** *The expectation of a random variable is defined as*

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

Definition 13 (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 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

□

1.9.1.2 Uncountable sets and infinitessimals

Conventional probability restricts Ω 's to only include sigma-algebras, that is, sets of which every subset is guaranteed to be measureable. This is neccessary because it turns out that some uncountable sets do indeed have subsets that you cannot assign a probability to. However, we can sidestep this difficulty by restricting ourselves only to countable sets. If we need continuity, we can include infinitessimals in that set.

1.9.1.3 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: $P(A|B) \neq P(B|A)$

1.9.2 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.

1.10 Statistics

1.10.1 Correllation

Assume an inner product space.

1.10.2 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

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

We can find the gradient of w by:

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

1.10.3 Principal component analysis

1.10.4 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.

1.11 Graph Theory

1.11.1 General properties

1.11.1.1 Degrees

Theorem 6. *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* $\sum_{V_G} \deg(v_n) = 2|E_G|$

1.11.2 Walks and paths

Definition 14. *A walk is any sequence of vertices that are connected by an edge.*

Definition 15. *A path is a walk where no vertex appears twice.*

Theorem 7. *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 8. *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.

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

Theorem 10. *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*

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

Theorem 12. *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 13. *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 14. *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 $N + L = E + 1$

□

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

□

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

$$M = \{Q, q_0, F, \Sigma, \delta\} \tag{1.9}$$

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:

$$\delta(q, \sigma) = q' \tag{1.10}$$
$$\delta : Q \cdot \Sigma \to Q \tag{1.11}$$
$$\Delta(\vec{\sigma}) = \delta(\delta(\delta(q_0, \sigma_0), \sigma_1), ...) \tag{1.12}$$

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

$$L(M) = \{\vec{\sigma} | \Delta(\vec{\sigma}) \in F\} \tag{1.13}$$

Since δ can be expressed as a graph, a version of theorem 10 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[3].

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.

[3]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:

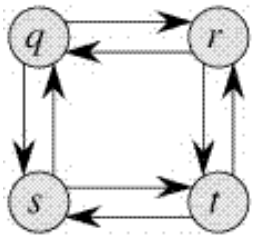
$$sol(para) = f(sol(para'), sol(para''), ...)$$

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

$$S = sol(para)$$

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

In practice we might instead want to partition S into several new sets S' and 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 Dynamic programming for hidden Markov-models (Viterbi algorithm)

1.14 Calculus

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

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 beleive 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 15 to prove the following:

Theorem 16. $\forall n, m \in \mathbb{N} : n : std \wedge m : nstd \to (n + m) : nst$

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

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

□

1.14.2 Limits

1.14.3 Sequences and series

1.14.3.1 Tailor

1.14.3.2 Fourier

1.14.3.3 Laplace

1.14.4 Eulers 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.

1.14.5 Integration

1.14.5.1 Integration strategies

u-substitution

Integration by parts is the last trick up our sleve when all other strategies havent 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:

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

[4]

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 sollution.

1.14.6 Vector calculus

inetration 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

[4] The proof goes like this: ...

1.15 Number theory and cryptography

1.15.1 Divisibility

Definition 16. *Divisability:* $a|b \leftrightarrow \exists k : ak = b$

Theorem 18. $a|b \wedge b|c \rightarrow a|c$

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

□

Theorem 20. $\forall a, b : \exists! q, r : a = qb + r \wedge 0 \leq r \leq q$

Theorem 21. *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.

1.15.2 Hashing

1.15.3 Encryption

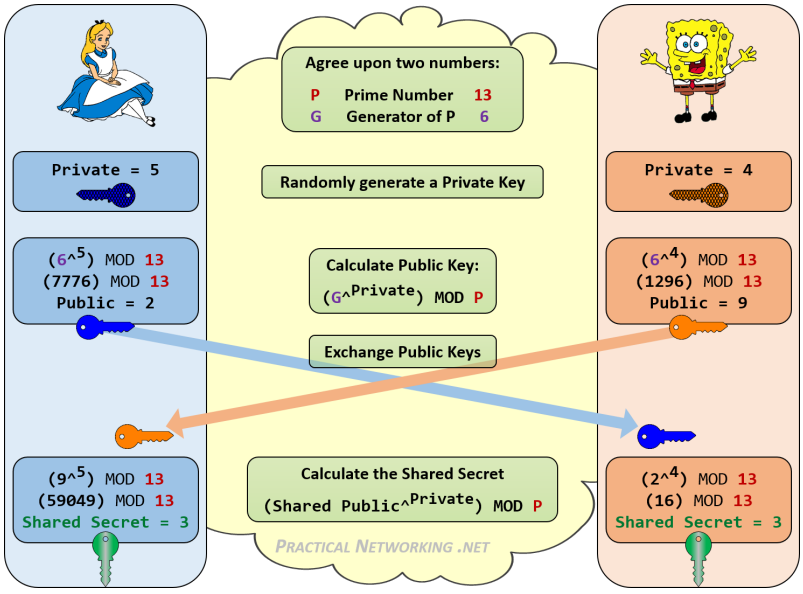
1.15.3.1 Symmetric encryption

1.15.3.2 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

1.16 Artificial intelligence

1.16.1 Categorisation

1.16.1.1 Neural networks

Backpropagation networks There are two ways in which we can derive the backpropagation-algorithm. One is purely analytical, the other bayesian. We'll demonstrate both.

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 $e = \frac{1}{2}\sum_t (\vec{y}_t^* - \vec{y}_t^L)^2$

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:

$$\frac{de}{d\mathbf{W}^l} = \left(\frac{de}{d\vec{x}^l}\right)^T (\vec{y}^{l-1})^T$$

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.

$$\frac{de}{d\vec{x}^l} = \delta^l \odot f'(\vec{x}^l)$$
$$\frac{de}{d\mathbf{W}^l} = \left(\frac{de}{d\vec{x}^l}\right)^T \left(\vec{y}^{l-1}\right)^T$$
$$\delta^{l-1} = \frac{de}{d\vec{x}^l} \mathbf{W}^l$$

Gaussian processes The bayesian derivation of the backpropagation algorithm gives rise to the interesting idea that backprop-networks may be generalized into ...

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.

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.

[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.

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:

$$\frac{de}{d\vec{x}^l} = \frac{de}{d\vec{y}^l} \frac{d\vec{y}^l}{d\vec{x}^l} = \delta^l$$

$$\frac{de}{d\vec{w}^l} = 0$$

$$\delta^{l-1} = \frac{1}{4} \frac{de}{d\vec{x}^l}$$

1.16.2 Computer vision

Feature detection and pose estimation Say you want to locate the position of the nose in a portrait.

1.16.3 Symbolic AI

Contrary to the before mentioned approaches, symbolic AI uses logical deduction instead of numerical processing to arrive at decissions. If neural nets and decision-trees learning from data can be called building *experince*, 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: can interpret the symbols $\forall$ and $\exists$ and properties
- 2nd order logic: understands relations: can interpret the symbol $\in$
- higher order logic: is better at inference; can create new if-then-statements as a result of inference or even explore

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 existance 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 Programm 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

$$\Theta(g(n)) = \{f(n) | \exists c_1, c_2, n_0 : \forall n \geq n_0 : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

Simmilarly, 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 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)
```

2.1.7 Dynamic programming

A natural extension to the idea above here is dynamic programming. There we also make use of the fact that some instructions can be simplified if we remember the result of the previous instruction.

2.1.8 Solving recurrences

Sometimes we dont have to do a recursive sollution 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 examplify some methods for finding the closed form expression.

2.1.8.1 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 sollution**. Also, if a root r occures v times, $r^n, nr^{n-1}, n^2r^n ..., n^{v-1}r^n$ are also sollutions.

4. Also, every linear combination of the above is also a sollution. So, a sollution 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 sollution**.

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 sollution from step 4 above.

2. Find a **particular sollution** to the equation including $g(n)$.

3. homogeneous solltution + particular sollution = **general sollution**

4. Now for the general sollution, 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
```

$Q(i)$ **:** statement

Initial state $Q(0)$ **:** sometext

Transition $Q(i) \to Q(i+1)$ **:** sometext

Final state $Q(n)$ **:** sometext

2.2.2 Matrix inverse

Gauss Jordan

2.2.3 Polynomals

In this section, we will deal with the polynomal $P_A(x) = 1x^3 + 2x^2 + 3x + 4$. Working with polynomals turns out to be quite important in everyday practice - especially when you're working with generating functions.

2.2.3.1 Coefficient-representation of polynomals

The most common representation of this polynomal is its coefficient representation $[4, 3, 2, 1]$. This representation is very convenient for fast evaluation at any x using Horner's method:

```
#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;
}
```

2.2.3.2 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):

```
#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;
}
```

2.2.3.3 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(l, 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;
}
```

2.2.3.4 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 $(1^{1/4})_1, (1^{1/4})_2, (1^{1/4})_3, (1^{1/4})_4 = 1, i, -1, -i$.

2.3 Data-structures

2.3.1 Stack

Stacks perform push and pop in O(1). However, they do search in O(n).

```c
#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.

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

General computer theory

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.

- PCI
- SATA
- USB
- Firewire

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.

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 thousends 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. 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 ammount 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 frequencies at which devices may communicate:

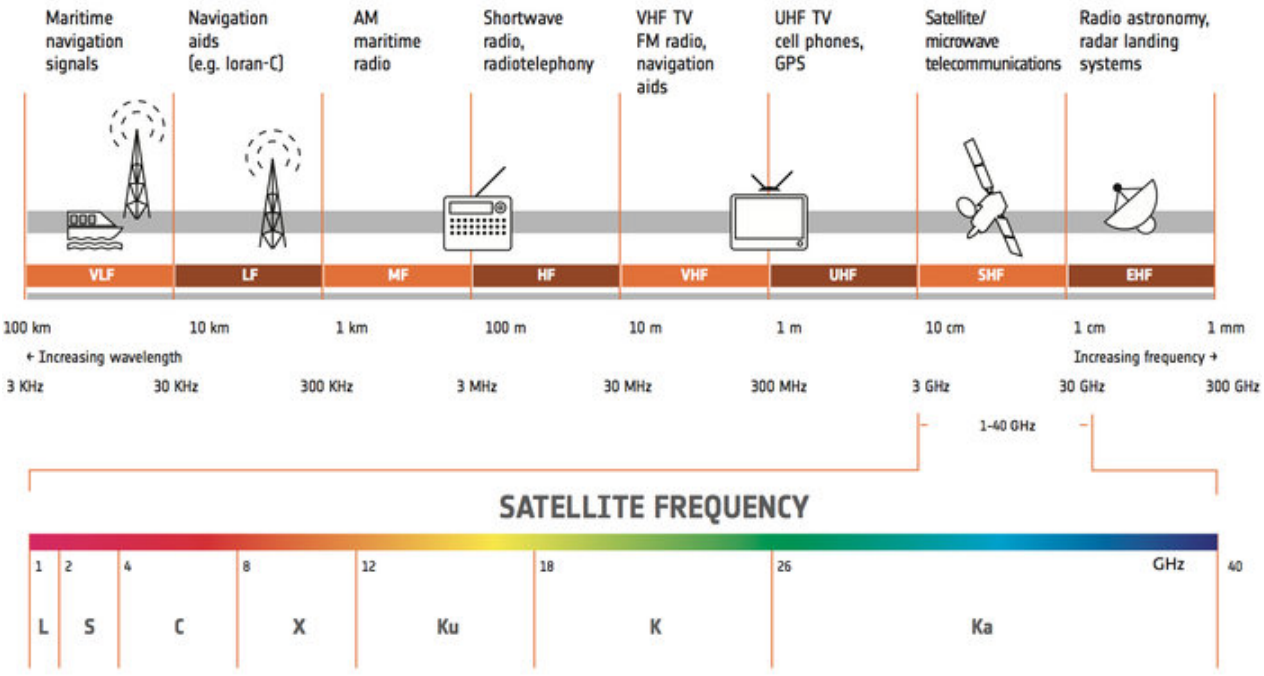
- 2.4 GHz: slower, further reach
- 5.0 Ghz: faster, but only at shorter distance

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 Cookies

Since http is a stateless protocol, we need some helper to store state over several page-visists.

Cookie-domain : A cookie is stored on the browser and only passed allong to the server with an http-request if the requested url matches the cookie-domain.

3.4 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.

We need to serve multible needs:

- Programmer needs: consisten API that abstracts away the details of the data-storage. A programmer should not need to know on what server the requested data is stored (except when we want to use that detail-knowledge to get better performance).
- Business needs:
  - consistency: a request mustn't result in different responses
  - availability: your requests must always be served
  - low latency: a response should come fast

Operations on a single machine have a total order, on distributed machines a partial order transition functions: 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:

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

3.4.1 Important theorems

3.4.1.1 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.

3.4.1.2 CAP-Theorem

3.4.2 Important alogrithms

3.4.2.1 one-, two- and tree-phase commit

3.4.2.2 paxos algorithm

3.4.2.3 raft algorithm

3.4.2.4 map reduce algorithm

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

3.4.3 Databases

3.4.3.1 ACID

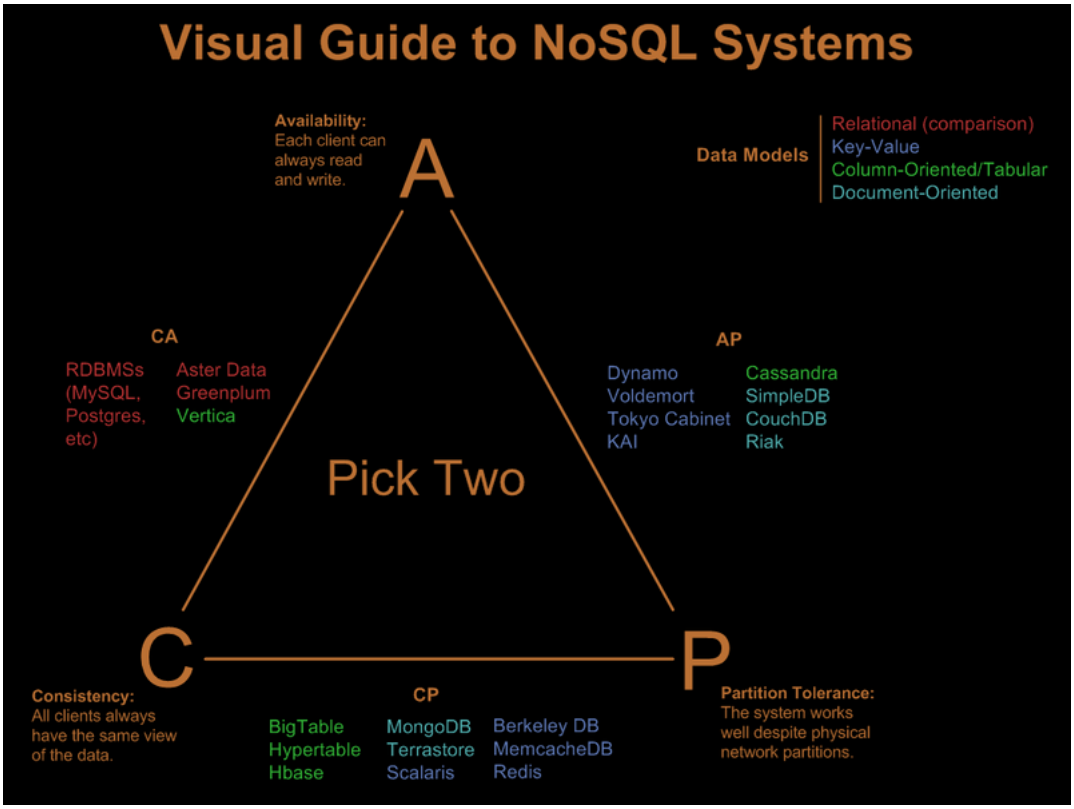
A transaction is a sequence of database operations that can be considered one independent, coherent unit of work. Database transaction should have the ACID-properties:

- 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.

3.4.3.2 NoSQL Database types



3.4.3.3 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:
                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
```

3.4.4 Distributed cache: redis

3.5 Virtualisation

3.5.1 KVM/QEMU

3.5.2 Docker

3.6 Sound

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.

3.6.1 Audio Standards

Raw audio consists of `short[]` arrays containing raw amplitude data.

Midi comes in two forms: JACK-Midi and ALSA-Midi.

3.6.2 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.

3.6.3 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.

3.6.4 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.

3.7 Security

Chapter 4

Programming languages

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 compuiling and linking.

- **Compiling your own code** The compiler generates an object file from your sourcecode. An object-file is a compiled piece of code together with some meta-information on what kind of functions and structures it containes.
- **Linking in libraries** After that, the include-directories are scanned for libraries that contain implementations of the header-files you included in your sourcecode. 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 programm. 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
```

```
# Compilen
#   -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

# Linken
#       -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 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.

4.3.1 General

4.3.1.1 Basic theory

4.3.1.2 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

4.3.2 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

4.3.3 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.

4.3.4 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);
}
```

4.3.5 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());
        }
}
```

4.3.6 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).

4.3.6.1 CSV to POJO and back: Bindy

Important annotations in the pojo are:

4.3.6.2 JSON to POJO and back: Jackson

Important annotations in the pojo are:

4.3.6.3 XML to POJO and back: JAXB

Important annotations in the pojo are:

4.3.7 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.

4.3.7.1 Extracting from xml: XPath

4.3.7.2 Extracting from json: JsonPath

4.3.8 Databases

Java allows for simple query-based database access (JDBC) and for automated marshaling of pojos into tables (JPA and hibernate).

4.3.8.1 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()){
    ...
}
```

4.3.8.2 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 generalised 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.

4.3.9 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.

4.3.9.1 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;
       }
   }
```

4.3.9.2 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);
            }
}
```

4.3.9.3 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

4.3.10 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.

4.3.10.1 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`

4.3.10.2 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.

4.3.10.3 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.

4.3.10.4 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.

4.3.11 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.

4.3.11.1 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";
        }
}
```

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>
```

[2]Its good to know that there is a popular alternative to jersey known as restlet.

```
        <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>
```

4.3.11.2 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 ....

Then you feed it your beans like this:

....

4.3.11.3 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>
```

4.3.12 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:

[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

- 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`

4.3.13 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");
```

4.4 Groovy

Python was built as a scripting language with c-interoperability. This lets it access the fast, native c libraries, but also forces it to a certain platform dependence. To use modules like numpy or heroku, the relevant c-so's (math and ssh, respectively) must be installed on the system. Groovy was built as a scripting language with java-interoperability. Where python is a scripting-gateway into the native system, groovy is a scripting-gateway into java. This defines groovy's use-case: use groovy when you want to write a scripting-project that can be nicely packaged to be run anywhere where a jvm is installed - which is practically anywhere, including windows, a domain where python lacks somewhat.

4.4.1 Building, running, and packaging

Contrary to java, it is quite viable to build and run groovy/gradle from the command line, which allows us to skip an ide.

4.4.2 Groovy on grails

4.5 Go

4.5.1 Functional programming

4.5.2 Structures

4.5.3 Modules

4.5.4 Goroutines

4.5.5 Database access

4.5.6 Web access

4.6 Python

Python is a wonderful calculator. It is not intended for large projects, but for one-off scripts, prototypes, data-analyses and the like.

4.6.1 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()}
```

4.6.2 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
```

4.6.3 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 wrapper():
        print("This is printed before the function is called")
        someFunc()
        print("This is printed after the function is called")
    return wrapper

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()
```

4.6.4 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.

4.6.4.1 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]
```

4.6.4.2 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))
    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
```

4.6.4.3 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 """
```

[4]This is because internally, every class inherits from a default-class named 'type'

```
    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)
```

4.6.4.4 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)
```

4.6.5 Plotting

Matplotlib is a matlab inspired library for plotting.

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
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()
```

4.6.6 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
```

4.6.7 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
```

4.6.8 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`.

4.6.9 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)))
```

4.6.10 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.

4.6.11 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
```

4.6.12 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.

4.6.13 Regex

4.6.14 Packages

There are two ways of importing other scripts. The first is to just add the script location to the include path and then load the script like this:

```
import sys
sys.path.append('dependencies/pybuilder')
from buildtool import *
```

This, however, has a few disadvantages. .....

So it may be better to instead turn your project into a package. A package is merely a directory that contains the scripts together with a (potentially empty) file named `__init__.py`. With this, you can go like this:

```
import sys
sys.path.append('dependencies')
import pybuilder.buildtool as bt

help(bt.execute)
```

The init-file may however also contain some additional information.

4.6.15 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`

4.7 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".

```
; definitions are identical for variables and functions
(define name "Michael")
(define (add x y)
    (+ x y))

; use anonymous functions to create functions as return-values on the fly
(define (createPowerFunc pwr)
  (lambda (base)
    (expt base pwr)))
(define pwr4 (createPowerFunc 4))
(pwr4 3)
> 81


; conditionals
; if/else is simple:
(if (equal? name "Andreas")
      "Zieh ab!"
      "Servus!")
; if you want if/else with more than two cases, use cond:
(define (commentMyWeight w)
    (cond
      [(< w 50) "Gosh, you're a fether!"]
      [(< w 70) "Sounds fine."]
      [(< w 90) "You're a dude, right?"]
      [else "Seriously, thats too much!"])


; datatypes
; list
'("1" 2 c)
(list "a" "b")
```

4.8 PHP

Php is like a dynamically typed, scripted clone of java exclusively for webserver use.

4.8.1 Absolute and relative paths

You say your website is in `http://localhost/mywebsite`, and let's say that your image is inside a subfolder named `pictures/`:

Absolute path If you use an absolute path, `/` would point to the root of the site, not the root of the document: `localhost` in your case. That's why you need to specify your document's folder in order to access the pictures folder: `"/mywebsite/pictures/picture.png"` And it would be the same as: `"http://localhost/mywebsite/pictures/picture.png"`

Relative path A relative path is always relative to the root of the document, so if your html is at the same level of the directory, you'd need to start the path directly with your picture's directory name: `"pictures/picture.png"`

But there are other perks with relative paths:

dot-slash (./): Dot (.) points to the same directory and the slash (/) gives access to it. So this: `"pictures/picture.png"` Would be the same as this: `"./pictures/picture.png"`

Double-dot-slash (../) In this case, a double dot (..) points to the upper directory and likewise, the slash (/) gives you access to it. So if you wanted to access a picture that is on a directory one level above of the current directory your document is, your URL would look like this: `"../picture.png"`

In practice Let's say you're on directory A, and you want to access directory X.

```
- root
   |- a
      |- A
   |- b
   |- x
      |- X
```

Your URL would look either:
Absolute path: `"/x/X/picture.png"` *Relative path*: `"./../x/X/picture.png"`

4.8.2 XDebug

The profiler can be very helpful. Setup:

- Make sure xdebug is installed
- append this to your apache-php.ini:

```
        xdebug.profiler_enable=0
        xdebug.profiler_enable_trigger=1
        xdebug.profiler_output_dir=/home/hnd/freeForAll/
```

- append this to the url you want to profile: `&XDEBUG_PROFILE`.
- To render the resulting `cachegrind.out` file: `git clone https://github.com/jokkedk/webgrind`.

Note that the resulting file can be over 500 MB big.

4.8.3 Apache

4.8.4 Setting up tls

- Buy Certificate (or create a self-signed one for development). At the end you should end up with a certificate somwhere on your server, like `/home/hnd/tls/server.crt`.
- Configure Apache to resolve https-calls.

4.8.4.1 htaccess

4.8.4.2 IP based and name based resolution

IP based : this is for when more than one IP-adress points to your server. Then you can differentiate between different IP-adresses in your apache-configuration: `<VirtualHost 10.112.70.171:80>` and `<VirtualHost 10.112.70.172:80>`.

Name based : this is for when multiple DNS-records point to the one ip of your server. Then you can differentiate between these records using the `ServerName` pragma: `<VirtualHost *:80> ServerName www.hnd.bybn.de` and `<VirtualHost *:80> ServerName www.gkd.bybn`

4.8.5 Modes

4.8.5.1 mod php

4.8.5.2 CGI

4.8.6 Configuration

`/etc/php5/apache2/php.ini` . Additionally, all directives in the `php.ini` can be overwritten with

- `ini_get(string <directive name>)`: Retrieves current directive setting
- `ini_set(string <directive name>, mixed <setting>)`: Sets a directive

4.8.6.1 Apache config

Global config

Vhost config

4.8.6.2 Php config

global

CLI

apache

4.8.7 Logging

When coding java, there is a strict, by-default differentiation between your app's output and your error-output. Your app is displayed as is (stdout), your errors in the console (stderr). Since php-apps are usually deployed on a server, you can only acces stdout via a browser and stderr via the servers logs. Php-devs tend to use `echo` calls in their code to display logging information inside the stdout. This is actually very bad practice! Avoid doing this! Good practice is to write logging information to the logs using `error_log("<pre>".print_r($werte,true)."</pre>", 3, "/home/hnd/freeForAll/php_custom_error.log");` and keeping the error logs open at all times. Here are the logs you should keep open:

- ... : The apache-access-log.
- ... : The apache-error-log.
- ... : The php-error-log.
- Any custom log you specify with `error_log`.

The configuration of logs is handled in the `php.ini` with the following directives:

4.8.8 Build automation

We build, test and distribute php-projects using Jenkins. For this, we make use of a pipeline. The scirpt can be found here: http://hnd-jens.rz-sued.bayern.de:8181/applikation/jenkinsPipeline/blob/master/jenkinsfile.groovy

Table 4.3: Php logging directives (recommendations are for production server)

Directive	Default Setting	Recommend Setting	Description
*docref_root	1	0	An error format containing a documentation page reference
**display_errors	1	0	Defines whether error output is sent along with regular outpu
display_startup_errors	0	0	Whether to display PHP startup sequence errors
error_append_string	null	null	String to output after an error message
***error_log	null	“/path/to/log/file”	File path to a writable (by web server user) error log file. Th
error_prepend_string	null	null	String to output before an error message
error_reporting	null	E_ALL (or 32767)	Sets the error reporting level
log_errors	0	1	Defines whether application errors are logged
log_errors_max_length	1024	0	Sets the maximum byte length of log messages. “0” represent
ignore_repeated_errors	0	0	Do not log repeated messages
ignore_repeated_source	0	0	Ignore message source when ignoring repeated messages
report_memleaks	1	0	Reports memory leaks detected by the Zend memory manage
track_errors	0	1	Last error message available in $php_errormsg

4.8.8.1 Phpunit

Structure A *testsuite* is a group of *testcases*. Testcases[5] are the baseclass that every testclass must extend to get access to the different `assert` methods.

configuration When php-unit is invoked we can tell it to use a configuration-xmlfile instead of getting all its arguments from the commandline. We tell it where to look for that file by calling `phpunit --configuration tests/phpunit.xml`. Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>

<phpunit
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="https://schema.phpunit.de/6.3/phpunit.xsd"
     backupGlobals="true"
     backupStaticAttributes="false"
     <!--bootstrap="/path/to/bootstrap.php"-->
     cacheTokens="false"
     colors="false"
     convertErrorsToExceptions="true"
     convertNoticesToExceptions="true"
     convertWarningsToExceptions="true"
     forceCoversAnnotation="false"
     mapTestClassNameToCoveredClassName="false"
     printerClass="PHPUnit_TextUI_ResultPrinter"
     <!--printerFile="/path/to/ResultPrinter.php"-->
     processIsolation="false"
     stopOnError="false"
     stopOnFailure="false"
     stopOnIncomplete="false"
     stopOnSkipped="false"
     stopOnRisky="false"
     testSuiteLoaderClass="PHPUnit_Runner_StandardTestSuiteLoader"
     <!--testSuiteLoaderFile="/path/to/StandardTestSuiteLoader.php"-->
     timeoutForSmallTests="1"
     timeoutForMediumTests="10"
     timeoutForLargeTests="60"
     verbose="true">
  <!-- ... -->
</phpunit>
```

Autoload the file `--bootstrap autoload.php` is a php file that contains any php-code that needs to be executed before the actual tests are run. Usually it consists merely of a bunch of `require` statements. These are used to load the testable files as well as their testfiles.

Fixtures: setUp() and tearDown() ...

[5]Before the introduction of namespaces, these were known as `PHPUnit_Framework_TestCase`.

4.8.9 Drupal

4.8.9.1 Database

You can have drupal print out a query with `$query->__toString()` or just `print $query`. To also see the arguments, try `strtr((string) $query, $query->arguments());`.

4.9 Javascript

4.9.1 Prototype inheritance

```
var alien = {
  kind: 'alien'
}

var person = {
  kind: 'person'
}

var zack = {};

// assign alien as the prototype of zack
zack.__proto__ = alien

console.log(zack.kind); //=> alien

// assign person as the prototype of zack
zack.__proto__ = person

console.log(zack.kind); //=> person
```

Prototype lookups are dynamic. You can add properties to the prototype of an object at any time, the prototype chain lookup will find the new property as expected.

```
var person = {}

var zack = {}
zack.__proto__ = person

// zack doesn't respond to kind at this point
console.log(zack.kind); //=> undefined

// let's add kind to person
person.kind = 'person'

// now zack responds to kind
// because it finds 'kind' in person
console.log(zack.kind); //=> person
```

Javascript asserts that one usually doesn't need classes, since you most often only create one instance. If you do need to share things between two objects, add the shared things to a third object (the prototype) and give that prototype to the two objects.

```
var me = {name: "Michael"};
var you = {name: "Eva"};
var singer = {
    sing: function() { console.log("Me and you and a dog named " + this.name); }
};
me.__proto__ = singer;
you.__proto__ = singer;
me.sing();
you.sing();
```

4.9.2 Constructor functions

Honestly, I dislike constructor functions. Constructor functions are factories of objects. They have a `prototype` property. But watch out! `Robot.__proto__` is the prototype of the function `Robot`, whereas `Robot.prototype` is the prototype of the object that the function generates. In fact, every function in javascript has a `prototype` property, because there is no way of knowing which function is going to be used as a constructor function.

```
var Robot = function () {
    this.sayHi = function () {
        console.log("Bleep bloop.");
    };
};
var r2d2 = new Robot();

Robot.__proto__ // => 'native code'
Robot.prototype // => the prototype of r2d2
```

We can still use prototypical inheritance with the objects returned from a constructor function:

```
var Fighter = {
        fight: function () {
                console.log("Boom!");
        }
```

```
};

Robot.prototype.__proto__ = Fighter;

r2d2.fight();
```

4.9.3 Module Pattern

A module is created with a self-executing closure:

```
// buz.js
var buz = (function ($) {
    var buz;

    buz.sayHi = function () {
        console.log("hi!");
    };

    return buz;
} ($));
```

Ultimately, this can be a little hard to read. There are tools like browserify, however, that let you write your sourcecode locally like you would in node, and then bundle all your requires together into one single script.

4.9.4 Binding this

`this` refers to whoever calls a function.

```
var me = {name: "Michael"};
function printThis() { console.log(this);}
me.pt = printThis;
printThis(); // window
me.pt(); // me
```

Because this can get confusing, javascript allows us to `bind` an object to a function to always be the value of `this`.

```
var me = {name: "Michael"};
function printThis() { console.log(this);}
printThisBound = printThis.bind(me); // note that printThis is not changed in this process.
printThis();  // window
printThisBound(); // me
```

4.9.5 Node and NPM

Node is a webserver that you can program in javascript, npm is a javascript package manager.

Serving files with node

Creating projects with npm Initiate a project: `npm init`

Npm has already done most stuff in the `package.json` for you. But you still need to add the dependencies and the start-script like for example so:

```
{
  "name": "electron-quick-start",
  "version": "1.0.0",
  "description": "A minimal Electron application",
  "main": "main.js",
  "scripts": {
    "start": "electron ."
  },
  "repository": "https://github.com/electron/electron-quick-start",
  "author": "GitHub",
  "dependencies": {
    "electron": "^1.6.2"
  }
}
```

Fetch dependencies using `npm install`. You can add further dependencies to your `package.json` by calling `npm install --save-dev gulp`, where `--save-dev` means that we only want to use the `gulp` package during development, but not in the final, deployed product. Run the app with `npm start` (Which in turn only just calls "electron .").

Uninstall packages with `npm uninstall <package-name>`

Modules in npm A module is created with a pretty simple syntax. Contrary to the closure-module-pattern we'd use in web apps, in node there is no need for a closure. The only thing that gets exported is what we put in *module.exports*:

```
var myMod = {
    sayHi : sayHi
};
var sayHi = function () {
    console.log("hi!");
};
var privateMethod = function () {
    console.log("this method won't be visible outside of this script");
};
module.exports = myMod;
```

Once some value for *module.exports* has been set, it can be imported as such:

```
    // app.js
    var myMod = require('./myMod.js');
    myMod.sayHi();
```

Buildautomation If you have a really complex build-process, you might want to use something like gulp. However, for a simple setup, npm's `scripts` are more than enough.

```
{
  "name": "fstsktch",
  "main": "index.js",
  "scripts": {
    "devel": "watchify src/js/sketch.js -o build/main.js",
    "deploy": "npm run brwfy && npm run uglfy",

    "brwfy": "browserify src/js/sketch.js -o build/deleteMe/bundle.js",
    "uglfy": "uglify -s build/deleteMe/bundle.js -o build/main.js"
  },
  "devDependencies": {
    "browser-sync": "^2.23.6",
    "browserify": "^16.0.0",
    "uglify": "^0.1.5",
    "watchify": "^3.10.0"
  },
  "dependencies": {
    "p5": "^0.6.0"
  }
}
```

Your code will be automatically updated with `npm run devel`. It will be minified and deployed with `npm run deploy`.

Browser Traditionally, chrome and firefox are good choices for development, because they offer a nice developer console. In firefox, it can be tricky to avoid the js cache. Under settings, *disable* the field "HTTP cache bei offenem Werkzeugkasten deaktivieren". After doing so, the combination `ctrl - shift - r` will reload all js.

4.9.6 Var, let and const

- use `var` for a variable scoped inside this function.
- use `let` for a variable scoped inside this curly braces (this often a smaller scope than this function. Consider for example for-loops. Var will still exist after a for-loop, while let only exists inside it.)
- use `const` for a constant, that is, a variable that gets a value once and cannot thereafter be reassigned. (Works as expected when using primitives. But notice that in `const dog = age:3;` we can still change the age-property.)

4.9.7 Events versus Promises

Javascript doesn't do multithreading, but it does do asynchronous execution. Both events and promises are a tool that is meant to help you deal with asynchronous code.

Events are great for things that can happen multiple times on the same object—keyup, touchstart etc. Here is an example of how we would use events:

```
var img1 = document.querySelector('.img-1');

img1.addEventListener('load', function() {
  // woo yey image loaded
```

```
});

img1.addEventListener('error', function() {
  // argh everything's broken
});
```

4.10 Git

4.10.1 Lingo

- origin: default name for your remote repository
- head: *your* last *commit* on the current branch. Does not yet include non-commited changes.
  - When you switch branches with `git checkout`, the HEAD revision changes to point to the tip of the new branch.
  - When you do a `git pull`, your working directory may contain new stuff, which your head is lagging behind. To see the difference between the working dir and your last commit, do `git diff HEAD`
- index: The index is a single, large, binary file in baseOfRepo/.git/index, which lists all files in the current branch, their sha1 checksums, time stamps and the file name

4.10.2 Branches

Creating a new branch Don't forget to also specify the branches name when you push:

```
git branch schwebstoff_tabellen
git branch
git checkout schwebstoff_tabellen

# ... doing stuff ...

git push -u origin schwebstoff_tabellen
```

Checking out an existing branch Just first clone the whole repo and then switch to the branch:

```
git clone http://hnd-jens.rz-sued.bayern.de:8181/applikation/includes.git
git branch -a // also shows remote branches
git fetch origin schwebstoff_tabellen
git checkout schwebstoff_tabellen
```

Merging a branch back into master On the dev-branch, we first suck in all the changes that might have occured on the master while we were not looking. Once those changes have been properly integrated, we switch to the master and suck in the dev-branch.

```
// Step 0: safety measure: first get all new stuff from the master into your dev branch
git merge master

// Step 1: Go to master and insert all your dev stuff
checkout master
git merge schwebstoff_tabellen

// Step 2: cleanup
git branch --merged master // see all branches that have already been merged
git branch -d shwebstoff_tabellen  // don't need this branch any more
git push origin --delete schwebstoff_tabellen // also remove on remote

// Step 3: push
git push -u origin master
```

Merging remote changes on master into local master It can happen that somebody else has been messing with the master before you had a chance to commit your own changes. You will notice this by seeing your `git push` fail. In that case, try to first merge the remote changes into your local copy before pushing again.

```
git commit // first saving our latest changes
git fetch  // geting remote stuff in remote-tracking-branches. Doesn't change our files yet
git diff master origin/master // compares what you have in master vs. what happend to master remotely
git merge remotes/origin/master
```

This will try to merge the upstream changes into yours. If it fails, the conflicting file will contain annotations like these:

```
<<<<<<< HEAD
print("159 just edited this file")
=======
print("133 just edited this file")
>>>>>>> master
```

As you can guess, `<<<<<HEAD` is your code, `===` is a separator, and `>>>> master` is the end of the other guys code. Use a texteditor to resolve all conflicts. Eclipse actually has a graphical interpreter for the conflict-tags. Once the resolving is completed, a `git push` and `git commit` will complete the merge.

```
git status // shows that there is still a conflict
git add .
git status // shows that conflict is resolved, but merging still in progress
git commit -m "phew! managed to resolve the conflict!"
git push origin master
```

Comparing branches Diff can compare any commited branches:

```
// How is your branch different from the master?
git diff master..schwebstoff_tabellen
// What have you changed since your last commit?
git diff HEAD
```

Reverting and resetting is best explained in this post: https://stackoverflow.com/questions/4114095/how-to-revert-git-repository-to-a-prev
Basically, we have these options:

- `git revert <lastGoodCommitHash>..HEAD` will execute the inverse of all the operations and commit them as a new commit. This way, the old history stays intact.
- `git reset <lastGoodCommitHash>` will delete all the commits from `<firstBadCommitHash>..HEAD`. This way, you're rewriting history, which is really only good if you're working alone on a repo.

4.10.3 Creating your own repository

Basically, any git folder can be used as a repository. But you still need to do some work to expose that folder over the web. The easiest way to work is certainly over gitlab. For alternative means if you don't have a hoster, this site lists a few of the possibilities: http://www.jedi.be/blog/2009/05/06/8-ways-to-share-your-git-repository/.

4.10.4 Storing credentials

It is tedious to always have to type your name and password again. Here's how to avoid that:

```
git config [--global] user.name "Mona Lisa"
git config [--global] credential.helper "cache --timeout=3600"
```

```
git remote -v
git remote set-url origin https://github.com/USERNAME/REPOSITORY.git
```

Chapter 5

Stacks, platforms and libraries

5.1 Maps

A lot of this information comes from here: cite.opengeospatial.org The map-stack is certainly one of the most demanding stacks of technology that is in widespread use today, and clearly a field where a developer can sharpen his profile.

5.1.1 Available software

There are so many non-orthogonal tools out there that an overview can't do harm.

- Data-storage in db
  - Any PostgeSQL binding
  - Geodjango, for a Website-ORM.
- Map-creation
  - simple numpy raster data to geo-tiffs: gdal + basemap (deprecated) or cartopy. There is very good documentation here: annefou.github.io
  - QGis
  - MapServer (c)
  - GeoServer (java)
  - Mapnik (c, python)
  - TileMill (python)
- Serving data
  - QGis Server
  - MapServer (c)
  - GeoServer (java)
  - Mapnik (c, python)
  - FeatureServer (python): a WFS
- Displaying data
  - Folium: Builds websites based on Leaflet.js, but with a python api.
  - Geodjango with django-leaflet
  - OpenLayers

5.1.2 Types of services

Servers expose a number of services, commonly referred to as OWS. They are called via SOAP or REST-Parameters (KVP, for key-value-pairs, in geo-lingo). We'll only describe the GET-Parameters here. Here is an overview:

- WMS: delivers custom made images for a given map-square. Servers locally create GeoTiff files: high resolution images that are geo-referenced. They are then converted to png's and exposed via a HTTP-based API.
  - Version 1.1.1
    - GetCapabilites: `?service=wms&version=1.1.1&request=GetCapabilites`
    - GetMap `?service=wms&version=1.1.1&request=GetMap&src=EPSG:31468&format=image/png&BBOX=447,532,234,124&layers=l1,l2&widht=1000&hei`
    - GetFeatureInfo
  - Version 1.3
    - BBOX: andere Achsenreihenfolge! Von vielen clients noch immer nicht richtig verarbeitet.

- WMTS: has a pre-made pyramid of images. Clients pass an index for the pyramid instead of the coordinates of the selection.
  - Version 1.0.0.
    * GetCapabilities `?service=wmts&version=1.0.0&request=GetCapabilites`
    * GetTile `?service=wmts&version=1.0.0&request=GetTile&layer=l1&format=image/png&TileMatrixSet=default28m&TileMatrix=1&TileRow=1&Tile(`
    * GetFeatureInfo
- WFS: instead of images, a json-representation of features is passed to the client.
  –
- WPS

Usually, service providers expose *multiple* urls for you to query, so that you can fetch images from several servers in parallel.

5.1.3 Server: Geoserver

A popular choice for serverside software is MapServer (C, CGI and PHP) or GeoServer (Java). Geoserver is a kind of CMS for geodata. It is built with spring as its backend.

5.1.4 Configuring standard OWS

...

5.1.4.1 Creating custom OWS

You can create plugins that implement a certain type of OWS (WMS, WFS, WPS, or something completely custom) by building a jar that has geoserver as its maven-parent and then dropping the built jar into the geoserver-home-directory.

5.1.5 Client: OpenLayers

A popular javascript mapping library is openlayers.

- source: the ogc-protocol for how to *obtain* the data
  - `TileWMS`
  - `WMTS`
  - `XYZ`
- layer: the ol-graphics for how to *display* the data
  - `TileLayer`
  - `VectorLayer`

5.2 Android

5.2.1 ADB

It really makes a lot of sense to use a physical phone instead of an emulated one. To be able to communicate with your phone, however, you need the adb-server to be aware of the phone. There is a commandline-tool, `adb`, that you can use to have the adb-server talk to your phone. However, to be able to do that, you must first set up your phone accordingly.

- on your linux-machine, install `mtp-tools` and `mtpfs`.
- mount your phone using a *data* usb-cable
- enable developer-mode
- have your phone use the MTP protocol

With that given, you should be able to see your phone with `lsusb` and `adb devices`.

5.2.2 Basic concepts

The structure of an app is as follows: the compiled java-bytecode[1], togehter with all libraries and resources, is packaged into an APK (android application package), analog to a jar or a war.

The mainfest mostly tells the phone some meta-info about the APK; like what rights are required, what SDK version is required, what the main-activity is, and such.

A layout describes the app's appearence on one screen. Written in XML. The layout can be found in `res/layout`. Within a layout file, you can reference data from `res/values/strings.xml` by using the string `"@string/app\_someStringName"`.

An activity is a single, defined thing that your user can do. Activities are usually associated with one screen. In code, an activity is represented by a single class extending the class `Activity`. It largely has the role of *controller*, negotiating between the *view.xml* and a custom-class-backend. All activities need to be declared in the manifest.

An intent is a type of message that one activity can send to another. It can be used to start another activity within the same app or even in another app. It can even be broadcasted to all apps on the phone so that any app that might be able to handle the request may use it. Here is an example of how we might use an intent to switch to another activity:

```
letsGoButton = (Button) findViewById(R.id.startGameButton);
letsGoButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
                Intent goToGameActivity = new Intent(MainActivity.this, PlayActivity.class);
                startActivity(goToGameActivity);
        }
});
```

A view is a custom drawable thing, like buttons or menus. You create such a view when the predefined UI-elements don't suite your needs, like when you want to create an animation. You create a view by extending `View` and implementing `onDraw(Canvas canvas)`, `onTouchEvent(MotionEvent event)`, etc.

The file R.java is an automatically created file that android uses to keep track of the resources available to your app. For example, your activity will tell the phone to use the `activity_play.xml` layout file by going `setContentView(R.layout.activity_play);` inside the `onCreate` method. We can access things in `R` very much like we can in the `DOM`.

[1] actually, your code gets compiled to java-bytecode (i.e. several `.class` files), but from there is is compiled on to a single `.DEX` file to be run on androids version of the JRE, the ART.

5.2.3 Userinput

Android apps behave very much like html forms.

The steps are:

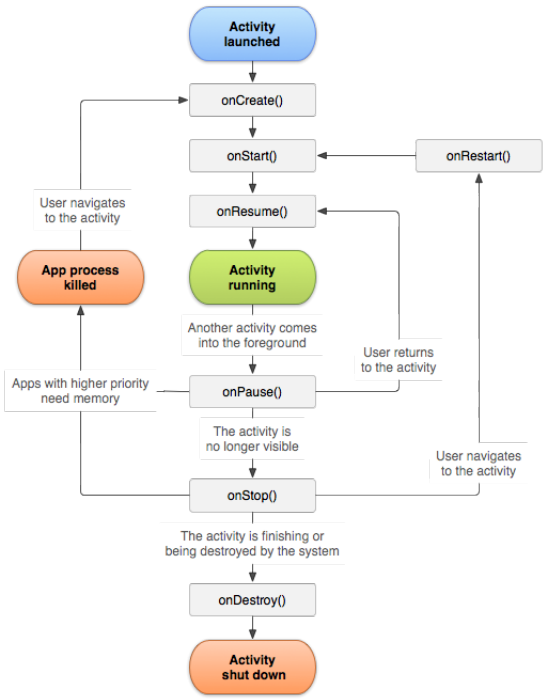
- The main-activity choses a layout (in this case, `activity_find_beer.xml`) that is to be displayed by using

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_find_beer);
}
```

- The layout specifies which method to call in the activity when a button is pressed. A button can have an attribute `android:onClick:"onClickFindBeer"`
- A click now is passed trhough to the activity, calling the method `public void onClickFindBeer(View view)`. In this case, `view` refers to the button that was pressed.
- Anywhere in the application logic, the activity can get a hold of any element of the layout by using `findViewById()` like this:

```
TextView brands = (TextView) findViewById(R.id.brands);
brands.setText("Gottle of geer");
```

5.2.4 Activities



This section handles activities in more detail.

5.2.5 Threading

Threads come in three different flavors:

- the main thread (aka. ui-thread)
  - listens for intents
  - listens for input (touch/speak) events
- the render thread

  - You don't normally interact with that thread. The main-thread passes it views to render, and the render thread executes the views `onDraw` methods.
  - However, you can steal work from it by creating a special kind of view: a `SurfaceView`. This kind of view exposes a canvas that is to be rendered in a custom thread.

- custom threads
  - simple custom threads extending `Thread`
  - android-specific custom threads extending `AsyncTask`

Handlers are a different thing than threads. They are not separate theads, they just postpone the execution of code. There basically promises: some code will be executed asynchronously, but sitll on the main thread. There are two usecases for handlers:

- You really want an action to happen on the main-thread, but at a later time. For example, ...
- Custom threads are not allowed to manipulate UI-elements on the main-thread. Instead, pass them a handler from the main-thread so they can call the handler. In other words: handlers are a (android-specific) method of inter-thread communication.

5.2.6 Drawing

There are three ways of drawing in android apps.

- When you only need a single animated item in an overal static UI, draw your animations into a view.
- When you want your drawing to be updated regularly, use a canvas.
- When you want to do 3d-rendering, use the OpenGL-API.

Using a custom view is certainly the most straightforward way of creating graphics. Consider this simple example:

```
public class PlayActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new FullScreenView(this));
    }
}
```

```
public class FullScreenView extends View {

    private Rect lilRect;
    private Paint painter;
    private static final int SQUARE_SIDE_LENGTH = 200;

    public FullScreenView(Context context) {
        super(context);
        lilRect = new Rect(30, 30, SQUARE_SIDE_LENGTH, SQUARE_SIDE_LENGTH);
        painter = new Paint();
        painter.setColor(Color.MAGENTA);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        canvas.drawRGB(39, 111, 184);
        canvas.drawRect(lilRect, painter);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        lilRect.left = (int) (event.getX() - (SQUARE_SIDE_LENGTH / 2));
        lilRect.right = (int) (event.getX() + (SQUARE_SIDE_LENGTH / 2));
        lilRect.top = (int) (event.getY() - (SQUARE_SIDE_LENGTH / 2));
        lilRect.bottom = (int) (event.getY() + (SQUARE_SIDE_LENGTH / 2));
        invalidate();
        return true;
    }
}
```

Using the canvas is a lot more direct than using a view-object. We still make use of a custom view, but instead of calling `invalidate()` when we're ready to draw, we host the game-loop inside the view and act upon the canvas directly.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        SceneLogic boardScene = new BoardScene();
        MainView mainView = new MainView(this, boardScene);
        setContentView(mainView);
        mainView.startRenderThread();
    }
}
```

```
/**
 * This class is there to present the SurfaceHolder as well as any touch- or system-events
 * to the RenderThread.
 */
public class MainView extends SurfaceView implements SurfaceHolder.Callback {

    private RenderThread renderThread;

    public MainView(Context context, SceneLogic sceneLogic) {
        super(context);
        SurfaceHolder surfaceHolder = getHolder();
        surfaceHolder.addCallback(this);
        renderThread = new RenderThread(context, surfaceHolder, sceneLogic);
    }

    public void startRenderThread() {
        renderThread.start();
    }

    public void stopRenderThread() {
        renderThread.setInactive();
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        return renderThread.getLogic().onTouch(event);
    }

    @Override
    public void surfaceCreated(SurfaceHolder surfaceHolder) {}

    @Override
    public void surfaceChanged(SurfaceHolder surfaceHolder, int i, int i1, int i2) {}

    @Override
    public void surfaceDestroyed(SurfaceHolder surfaceHolder) {}

}
```

```
public class RenderThread extends Thread {

    private final long frameTime = 17;    // How long (in millisec) a frame may take
    private SceneLogic sceneLogic;
    private Context context;
    private SurfaceHolder surfaceHolder;
    private boolean running;

    public RenderThread(Context context, SurfaceHolder surfaceHolder, SceneLogic sceneLogic) {
        this.context = context;
        this.surfaceHolder = surfaceHolder;
        this.running = true;
        this.sceneLogic = sceneLogic;
    }

    @Override
    public void run() {
        long updateDuration = 0;
        long sleepDuration = 0;

        while(running) {

            // Step 0 : how long did the loop take?
            long beforeUpdateRender = System.nanoTime();
            long delta = sleepDuration + updateDuration;

            // Step 1 : scene logic
            sceneLogic.update(delta);
```

```
            // Step 2: scene rendering
            Canvas canvas = surfaceHolder.lockCanvas();
            if(canvas != null) {
                sceneLogic.draw(canvas);
                surfaceHolder.unlockCanvasAndPost(canvas);
            }

            // Step 3: sleep for remainder of frameTime
            updateDuration = (System.nanoTime() - beforeUpdateRender) / 1000000L;
            sleepDuration = Math.max(2, frameTime - updateDuration);
            try{
                Thread.sleep(sleepDuration);
            } catch (Exception e) {
                e.printStackTrace();
            }


        }
    }

    public SceneLogic getLogic() {
        return sceneLogic;
    }

    public void setInactive() {
        running = false;
    }
}
```

5.3 C++ on android

https://www.youtube.com/watch?v=zOsUZjWbudI&list=PL0C9C46CAAB1CFB2B&index=3

5.4 VR on android

5.4.1 Basic app

The manifest should basically be a variant of this:

```
<manifest ...
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    ...
    <uses-sdk android:minSdkVersion="19" android:targetSdkVersion="24"/>
    ...
    <uses-feature android:glEsVersion="0x00020000" android:required="true" />
    <uses-feature android:name="android.software.vr.mode" android:required="false"/>
    <uses-feature android:name="android.hardware.vr.high_performance" android:required="false"/>
    ...
    <application
            android:theme="@style/VrActivityTheme"
            ...
        <activity
                android:name=".TreasureHuntActivity
                ...
                android:screenOrientation="landscape"
                android:configChanges="orientation|keyboardHidden|screenSize|uiMode|navigation"
                android:enableVrMode="@string/gvr_vr_mode_component"
                android:resizeableActivity="false">
                ...

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
                <category android:name="com.google.intent.category.CARDBOARD" />
                <category android:name="com.google.intent.category.DAYDREAM" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Gradle handles the specific dependencies that are required to make a 3d-app. At least you're going to need:

```
allprojects {
    repositories {
        jcenter() // this is where we get the vr libraries
    }
}
model {
    android {
        ...
        buildTypes {
            release {
                minifyEnabled true
                // Progueard minifies the APK. This line makes sure it doesn't obfuscate vr-libs.
                proguardFiles.add(file('../../proguard-gvr.txt'))
            }
        }
    }
}
dependencies {
    classpath 'com.android.tools.build:gradle:2.3.3'
    compile 'com.google.vr:sdk-base:1.120.0'
    compile 'com.google.vr:sdk-audio:1.120.0'
}
```

The layout in a vr-app is very simple. There is only one element to use: the `com.google.vr.sdk.base.GvrView`element.

The main task should in most case extend `GvrActivity`. This way, your activity gets access to a load of VR events.

5.5 Angular

- Module: lists all components (`declarations`) and all dependencies (`imports`)
- Component: Defines a custom html-Tag
- Service: accesses a server

5.5.1 Components

A component consists of a html-template and a ts-class. The template can access the classes properties and methods through directives. These directives are special syntax like `{}` or `*ngIf` that contain a string of ts-code that can be evaluated in the context of the class. It makes sense to think of a component as a way to view your model, not as directly related to the model itself. For example, if your model is a joke, you might have the components JokeDetail, JokeList, and JokeAdd.

- ts-file: has a method for every lifecycle-hook and a property for every bindable data.
- html-file:
  - interpolation: curly braces can read a classes properties. `<li>{{hero.name` uppercase}}¡/li¿—
  - input-binding: round braces capture input-events. `(click)="onSelect(hero)"` Captures an input and directs to a method
  - value-binding: square braces write to a classes properties. `[hero]="chosenHero"` assigns an object to a property.
  - two-way-binding: for reading *and* writing properties. `[(ngModel)]="hero.name"`, requires `FormsModule`
  - loops: `*ngFor="let hero of heroes"`
  - conditionals: `*ngIf="!selected"`

  Together, these might make a code section like this: `<li *ngFor="let hero of heroes"(click)="onSelect(hero)">{{hero.id}} -- {{hero.name}`

5.5.1.1 Template directives: details

***ngFor**

***ngIf**

5.5.2 Directives

Directives are essentially components without a template.

5.5.2.1 Attribute-directives

Instead of exposing a new custom html-element (as components do), attribute-directives expose a new custom *attribute* that we may add to any html-element. Here is an example of an attribute-directive in its simplest form: . Very often, you will want to create attribute-directives that fire events. This is where the `@Output` annotation comes into play. See this example:

```
import { Directive, Output, EventEmitter, ElementRef, HostListener } from '@angular/core';


@Directive({
  selector: '[guClickOutside]'
})
export class ClickOutsideInsideDirective {

  @Output() guClickOutside = new EventEmitter<MouseEvent>();
  @Output() guClickInside = new EventEmitter<MouseEvent>();

  constructor(
    private elementRef: ElementRef
  ) { }
```

```
    @HostListener('document:click', ['$event'])
    public onDocumentClick(event: MouseEvent): void {
      const targetElement = event.target as HTMLElement;
      if (targetElement && !this.elementRef.nativeElement.contains(targetElement)) {
        this.guClickOutside.emit(event);
      } else {
        this.guClickInside.emit(event);
      }
    }
  }
```

This directive can be used inside any component like so:

```
<gu-moveable
    [xPos]="xPos" [yPos]="yPos"
    ClickOutsideInsideDirective (guClickOutside)="onClickOutside($event)"
    ClickOutsideInsideDirective (guClickInside)="onClickInside($event)"
> ...
```

5.5.3 Routing

Routing is usually done in a separate module. It consist of two parts:

- The route definition, consisting of a path and the component that is to be displayed under that path.
- The html-links

As a side effect, when you decide to use routes, not all components need to be subcomponents of `app-component` anymore.

Configuring routing requires three steps. First, configure the routing in your `app.componnent.ts`:

```
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  {path: "", component: JokeListComponent},
  {path: "details/:id", component: JokeComponent},
  {path: "overview", component: JokeListComponent},
  {path: "add", component: JokeAddComponent}
];

@NgModule({
  imports: [
    BrowserModule,
    HttpClientModule,
    RouterModule.forRoot(routes)
  ]
})
export class AppModule { }
```

Second, change your `app.componnent.html` to include the router-outlet:

```
<h1>Welcome to {{title}}!</h1>
<router-outlet></router-outlet>
```

Third, make use of the routing:

- creating a reference: `<a routerLink="details/{{joke.id}}">{{joke.title}}</a>`
- getting data from a reference:

```
import {ActivatedRoute} from "@angular/router";

constructor(private route: ActivatedRoute) {
    let id = this.route.snapshot.paramMap.get('id');
}
```

Or, if you want to continue observing the url:

```
import {ActivatedRoute} from "@angular/router";

constructor(private route: ActivatedRoute) {
    this.route.params.subscribe( params => console.log(params) );
}
```

5.5.4 Modules

You create modules when you want to create some functionality that might be shared via npm. Modules are always imported into angular in the form of a class. To import your module into a site,

- download the module with npm
- `ng generate module <yourmodule>`
- add the module to the app-modules `import`s
- in `<yourmodule>.module.ts`,
  - create a class `<Yourmodule>Module`
  - configure that class with `@ngModule`
  - export the class

5.5.5 Services

Services are data providers. They do not have to call REST-interfaces. They might call a database or a file, or they might be used as a messaging service between components, or they might be a way for all your components to access the global game-state. But if they do happen to call a REST-service, they typically return an observable.

5.5.6 Component interaction

Usually, components will interact with each other by using a service, especially then when some shared data is held by that service (like state, a database or simmilar). Sometimes however, it makes sense to let components access each other more directly, especially when we're dealing with a parent-child relationship.

5.5.6.1 Subcomponents obtaining data from parent-template through Input

Subcomponents can obtain the values of their properties from supercomponents.

A component-class can have properties annotated with `@Input`. That means that the value of this property will be given through the components html-tag. For example, the component `Heroes` might have this tag in its template:

```
    <li *ngFor="let chosenHero of heroes">
        <app-hero-detail [hero]="chosenHero"></app-hero-detail>
    <li>
```

This `chosenHero` would be bound to the `hero` property in the `HeroDetail` component.

```
export class HeroDetailComponent implements OnInit {

  @Input()
  hero: Hero;

  constructor() {}
  ngOnInit() {}
}
```

5.5.6.2 Child notifying parent through EventEmitter

Children may annotate some properties with `@Output`.

```
@Component({
    selector: 'app-voter',
    template: `
        <h4>{{name}}</h4>
        <button (click)="vote(true)"  [disabled]="didVote">Agree</button>
        <button (click)="vote(false)" [disabled]="didVote">Disagree</button>
    `
})
export class VoterComponent {
    @Input()  name: string;
    @Output() voted = new EventEmitter<boolean>();
    didVote = false;

    vote(agreed: boolean) {
        this.voted.emit(agreed);
        this.didVote = true;
    }
}
```

With the `@Output` annotation, a new input-binding is available for the parents template: `(voted)`.

```
@Component({
    selector: 'app-vote-taker',
    template: `
        <h2>Should mankind colonize the Universe?</h2>
        <h3>Agree: {{agreed}}, Disagree: {{disagreed}}</h3>
        <app-voter *ngFor="let voter of voters"
        [name]="voter"
        (voted)="onVoted($event)">
        </app-voter>
    `
})
export class VoteTakerComponent {
    agreed = 0;
    disagreed = 0;
    voters = ['Mr. IQ', 'Ms. Universe', 'Bombasto'];

    onVoted(agreed: boolean) {
        agreed ? this.agreed++ : this.disagreed++;
    }
}
```

5.5.6.3 Supercomponents calling child-methods directly through ViewChild

On the other hand, sometimes a component needs access to it's children. A component can access its child-components with the `@ViewChildren` property-annotation.

```
import { CountdownTimerComponent }  from './countdown-timer.component';

@Component({
  selector: 'app-countdown-parent-vc',
  template: `
  <h3>Countdown to Liftoff (via ViewChild)</h3>
  <button (click)="start()">Start</button>
  <button (click)="stop()">Stop</button>
  <div class="seconds">{{ seconds() }}</div>
  <app-countdown-timer></app-countdown-timer>
  `,
  styleUrls: ['../assets/demo.css']
})
export class CountdownViewChildParentComponent implements AfterViewInit {

  @ViewChild(CountdownTimerComponent)
  private timerComponent: CountdownTimerComponent;

  seconds() { return 0; }

  ngAfterViewInit() {
    // Redefine `seconds()` to get from the `CountdownTimerComponent.seconds` ...
    // but wait a tick first to avoid one-time devMode
    // unidirectional-data-flow-violation error
    setTimeout(() => this.seconds = () => this.timerComponent.seconds, 0);
  }

  start() { this.timerComponent.start(); }
  stop() { this.timerComponent.stop(); }
}
```

5.5.7 Forms

Forms are a difficult subject in any framework or library. Angular offers to approaches to handling forms, template-driven (simple) or reactive (larger).

5.5.7.1 Template driven forms

These require little code, but they have a rather cluncky syntax.

5.5.7.2 Reactive (f.k.a. model-driven) forms

```
<form [formGroup]="profileForm" (ngSubmit)="onSubmit()">
    <label>
        First Name:
        <input type="text" formControlName="firstName">
      </label>
```

```
      <label>
          Last Name:
          <input type="text" formControlName="lastName">
      </label>
      <button type="submit" [disabled]="!profileForm.valid">Submit</button>
</form>
```

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl, Validators, ValidatorFn } from '@angular/forms';

@Component({
      selector: 'app-profile-editor',
      templateUrl: './profile-editor.component.html',
      styleUrls: ['./profile-editor.component.css']
})
export class ProfileEditorComponent implements OnInit {

      profileForm: FormGroup;

      constructor() { }

      ngOnInit() {

        // An example of a custom validator
        let nameMustNotBeBobValidator = function (): ValidatorFn {
          return (control: AbstractControl) => {
            let isValid =  control.value != "Bob";
            if(isValid) return null;
            else return {"MustNotBeBob": "The name must not be Bob!"};
          };
        };

        // the form
        this.profileForm = new FormGroup({
          firstName: new FormControl('First name',
            [Validators.required, Validators.minLength(4), nameMustNotBeBobValidator ]
          ),
          lastName: new FormControl('Last name')
        });
      }

      onSubmit() {
        console.warn(this.profileForm.value);
      }

}
```

5.5.8 Angular for maps

Angular's modular design allows us to create a declarative map-API, one where the whole map including markers is defined using html. We let ourselves be inspired by the `@agm` module, which implements google-maps bindings for angular.

5.5.9 RxJs: General concepts

As a singlethreaded, event-based environment, javascript has a bunch of event-based structures built in.

Observables are objects that maintain a list of subscribers. The subscribers must implement the subscriber-interface, which means that they must have methods `next`, `completed`, `error`. The "notify-all-subscribers" part is handled by an abstract base class from `RxJS`. To create an observable, you only need to pass in a function that tells the observable how to get its data.

There are multiple types of observables: Promise: - lets you wait for data - however, data is delivered in a single batch - there is no `observer.onCompleted()` method observable: - vs promises: - observables are like promises, but they can emmit multiple calls to `observer.onNext(data)`, and are cancelable and unsubscribeable. - vs subjects: - the next method is executed for every subscriber individually as soon as the subscriber subscribes. - observable cannot subscribe anywhere subject: - the next method is executed at some point, but not neccessarily when a subscriber subscribes. all subscribers receive the same data. - subjects implement both Obeservable and Observer interface, so they can also subscribe to other Observables if they want to. - `next` is called from outside the subject, instead of from inside like it is the case with observables. This makes that basically anyone can trigger a broadcast. behaviour subjects: - are subjects that have an initial state.

Subjects are like radio stations: you can tune in at any time, but from that moment on everybody gets the same data.

```
let radio = new Subject();

let listener1 = {
    onNext: (nextVal) => console.log("listener1 is hearing: " + nextVal),
    onCompleted: () => console.log("completed"),
    onError: (error) => console.log(error)
};
let session1 = radio.subscribe(listener1);

radio.next('Hello! Youre listening to ...');
// sleep

let listener2 = {
    onNext: (nextVal) => console.log("listener2 is hearing: " + nextVal);
}
let session2 = radio.subscribe(listener2);

radio.next('... RxJs radio!');
// sleep

session1.unsubscribe();
session2.unsubscribe();
```

Obeservables are like signing a rent-contract: depending on at what time you sign, your rent may be higher.

```
let landlord = Rx.Observable.create((tenant) => {
    while(true) {
        let currentRent = Math.random();
        tenant.onNext(currentRent);
        // sleep
    }
});


let observer1 = {
    onNext: (nextVal) => console.log(nextVal),
    onCompleted: () => console.log("completed"),
    onError: (error) => console.log(error)
}

let observer2 = {
    onNext: (nextVal) => console.log("blub!");
}

let contract1 = landlord.subscribe(tenant1);
// sleep
let contract2 = landlord.subscribe(tenant2);
// sleep

contract1.unsubscribe();
// sleep
contract2.unsubscribe();
```

These subtle differences yield different use-cases:

- To implement a clock, you want to use an observable, that sends you the current time and then an update every n milliseconds.
- To implement a game-loop, you want to use a subject, that sends the call to do `onUpdate` to all observers at the same time.
- To implement a chat-app you want to have a chat-app, where everyone can trigger a `next` when they hit enter.

5.6 Tensorflow

5.6.1 Low-level API

Working with tf always entails a two-step process. First, define a graph, and second, execute it.

```
import tensorflow as tf

# Step 1: define a graph
a = tf.Variable(initial_value=2) # Tensors are immutable by default. We must wrap them in a Variable to allow them to cha
b = tf.Variable(initial_value=3)
c = a * b

# Step 2: execute graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    result = sess.run(c)
    print(result)
```

Running a tensor means evaluating it - in the case of `c`, that means calculating `a * b`. There are two things that can be run by a session: tensors (to be evaluated) and operations (to be executed).

Tf is declarative: `c = a*b` does not return the result of the multiplication of a and b, but a tensor that is still waiting to be evaluated. This can make debuggung pretty hard, though. For debugging, we'd like tf to behave more imperatively, where every expression is immediately evaluated. We can acchieve this by using an interactive session, where every tensor can be evaluated immediately by calling `eval()`:

```
    sess = tf.InteractiveSession()
    c.eval()
```

Optimizers are a very common example of operations, so we show a very simple example of their usage.

```
    import tensorflow as tf
    import numpy as np
    import matplotlib.pyplot as plt


    N = 100
    a = 0.2
    b = 5
    xs_training = np.random.random(N) * 100
    ys_training = a*xs_training + b + np.random.random(N)
    xs_validation = np.random.random(N) * 100
    ys_validation = a*xs_validation + b + np.random.random(N)


    a_tf = tf.Variable(0.3421)
    b_tf = tf.Variable(0.41)
    xs_tf = tf.placeholder(tf.float32, shape=[N])
    ys_tf = tf.placeholder(tf.float32, shape=[N])
    model_tf = a_tf * xs_tf + b_tf
    loss_tf = tf.reduce_sum((model_tf - ys_tf)**2)
    optimizer = tf.train.AdamOptimizer(learning_rate=0.1)
    training_operation = optimizer.minimize(loss_tf, var_list=[a_tf, b_tf])


    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        for i in range(1000):
            sess.run(
                training_operation,
                {xs_tf: xs_training, ys_tf: ys_training}
            )

        prediction = sess.run(
            model_tf,
            {xs_tf: xs_validation}
        )


    plt.scatter(xs_validation, prediction)
    plt.scatter(xs_validation, ys_validation)
    plt.show()
```

5.6.2 Even lower level, and understsanding TF internals

Tensorflow is an automatic differentiation engine. That is not quite the same as a symbolic differentiation engine. The difference becomes appparent when you create a custom operation to add to TF. To do this, you need to:

- register a new operation in c++
  - operation-name
  - inputs including their shapes
- implement the operation in c++
  - operation
  - gradient of operation for all inputs
- create a python wrapper for the public python-tf-api (not really neccessary, because tf creates a default wrapper automatically)

So contrary to a symbolic engine, you have to explain to tf by hand how to calculate a gradient.

5.6.3 High-level API

When using keras, you usually don't see the low-level api at all - no tensors, no sessions, no `run`.

```
import numpy as np
import tensorflow as tf
import tensorflow.keras as k
import matplotlib.pyplot as pyplot


# 1. Data
N = 100
inputs = []
outputs = []
for n in range(N):
    i1 = np.random.randint(0,2)
    i2 = np.random.randint(0,2)
    o = 1 * ((i1 + i2) == 1)
    inputs.append([i1, i2])
    outputs.append([o])
inputs_np = np.array(inputs)
outputs_np = np.array(outputs)


# 2. Model
model = k.Sequential(layers=[
    k.layers.Dense(2, activation=k.activations.sigmoid),
    k.layers.Dense(1, activation=k.activations.sigmoid)
])

model.compile(
    optimizer=tf.train.AdamOptimizer(learning_rate=0.1),
    loss=tf.losses.mean_squared_error
)

# 3. Running

model.fit(x=inputs_np, y=outputs_np, epochs=1000)

result = model.predict(np.array([[0,0], [0,1], [1,0], [1,1]]))

print(result)
```

5.6.4 Toppics

5.6.4.1 Convolutional nets

```
    import tensorflow as tf
    import tensorflow.keras as k
    import matplotlib.pyplot as plt
    import numpy as np
    import dummydata as dd
    import utils as u


    # 1. Data
    imageWidth = imageHeight = 80
    batchSize  = 10
```

```
timeSteps = 10
#dd.createDummyDataset(batchSize, timeSteps, imageWidth, imageHeight)
data_in, data_out = dd.loadDummyDataset()



# 2. Net
model = k.Sequential([
    k.layers.Conv3D(5, (2,2,2),     name="conv3d_1",    input_shape=(timeSteps, imageWidth, imageHeight, 1)),
    k.layers.MaxPool3D(             name="maxpool_1"),
    k.layers.Conv3D(5, (2,2,2),     name="conv3d_2"),
    k.layers.MaxPool3D(             name="maxpool_2"),
    k.layers.Flatten(),
    k.layers.Dense(30,              name="dense_1"),
    k.layers.Dense(10,              name="dense_2"),
    k.layers.Dense(1,               name="dense_3")
])

model.compile(
    optimizer=k.optimizers.Adam(),
    loss=k.losses.mean_squared_error
)

model.fit(x=data_in, y=data_out, batch_size=batchSize, epochs=10)

model.predict(np.array([data_in[0]]))
```

5.6.4.2 LSTM

Lstm's can be tricky in that they need input in a very specific format.

```
import numpy as np
import tensorflow as tf
import tensorflow.keras as k



# 1. Data: sine-curve
nr_samples = 10
nr_timesteps = 4
nr_pixels = 1
inpt = np.zeros([nr_samples, nr_timesteps, nr_pixels])
outpt = np.zeros([nr_samples, nr_pixels])
for sample in range(nr_samples):
    for timestep in range(nr_timesteps):
        for pixel in range(nr_pixels):
            inpt[sample, timestep, pixel] = (np.sin(timestep * 0.1 + sample) + 1.0) / 2.0
    for pixel in range(nr_pixels):
        outpt[sample, pixel] = (np.sin((nr_timesteps + 1) * 0.1 + sample) + 1.0) / 2.0



# 2. Network: simple lstm
model = k.Sequential([
    k.layers.LSTM(1) # 1 = output dimension. for every timeseries we pass in, we want the one next value
])

model.compile(
    optimizer=tf.train.AdamOptimizer(learning_rate=0.1),
    loss=tf.losses.mean_squared_error
)

model.fit(x=inpt, y=outpt, epochs=500)

model.predict(
    np.array([[
       [0.5       ],
       [0.54991671],
       [0.59933467],
       [0.6477601 ]
    ]])
)
```

5.7 Google

5.7.1 IaaS: Google compute engine

This is a infrastructure-as-a-service.

5.7.2 PaaS: Google app engine

Here, you are given a VM and access to an enormous backend of services through APIs.

5.7.2.1 Google earth engine

5.7.2.2

5.7.3 Firebase

Contrary to GAE, in firebase, frontend-clients talk to a standard-api for the different services, not to your self-scripted Flask-server. However, there is still scripting: you can hook into the services' execution by using callbacks.

5.7.4 Google apps script

This is similar to what apple once did with applescript, but hosted on google cloud.

Chapter 6

Physics

6.1 Remote sensing

6.1.1 Electromagnetic radiation

6.1.2 Geospatial processing

6.1.2.1 Datatypes and protocols

Datacubes are tensors of (usually very large) (geo-)data.

OGC-WCPS: OGC's Web Coverage Processing Service is a language for filtering and processing multi-dimensional raster coverages.

Open-EO ia an API to speak to different geo-processing services like GEE. Unfortunately, open-EO stands in concurrence to WCPS.

6.1.3 Important satellites

6.1.3.1 Landsat

6.1.3.2 Sentinel

6.1.3.3 Modis

6.1.3.4 Copernicus

6.1.4 Important service providers

CHIRPS is ...

EODC : Earth Observation Data Center for Water Resources Monitoring

Eurac : Eurac is a private research company ...

Google Earth Engine provides ...

NASA's ECS (Earth observation center Core System) is a vast catalogue of ...

Chapter 7

Applications

7.1 Where should we meet?

n people want to decide at who's house to meet up. Does it matter? Can the sum of the paths that each person needs to travel vary? In fact, it can! Imagine that $n-1$ of them live in the same town, while the last of them lives far away on the other side of the country. If they all meet in the town, only one person has to travel very far, whereas if they meet on the other side of the country, $n-1$ people need to travel a large distance.

The distance to travel can be represented by a weighted connection matrix. Consider this example:

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 11 \\ 1 & 0 & 1 & 1 & 11 \\ 1 & 1 & 0 & 1 & 11 \\ 1 & 1 & 1 & 0 & 10 \\ 11 & 11 & 11 & 10 & 0 \end{bmatrix}$$

It is now trivial to see that we can find the optimal place to meet up as the column with the smallest sum - in this case this would be at the place of person d.

7.2 Effective wind on a ship

A ship can only take the force of the wind that is aligned with its own angle. To calculate the vector of the wind for the ship, we must project the actual wind-vector onto the ships direction-vector.

7.3 Making a touch-instrument

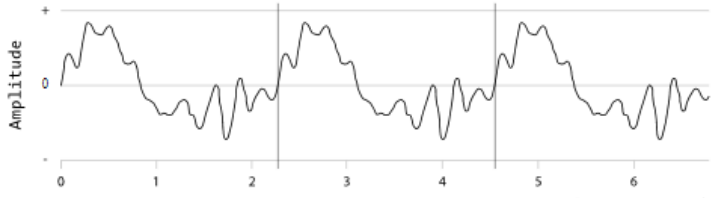
7.3.1 Music

7.3.1.1 Notes

A single tone x has only two atributes: frequency f_x and amplitude a_x a.k.a. pitch and loudness . The wave of a note is a function of these two: $w_x(t) = a_x \cdot \sin(f_x \cdot t)$

In practice, a note will stop being played at some time. Therefore, the real function for a note would be $w_x(t) \cdot s(t_1, t_2)$, where s is a step function. In FFT, however, we mostly treat a note as if it were a real periodic function - that is, never ending. We'll see details of this later.

By the way: The same note played on a piano still sounds different from a violin because physical instruments never play one pure note. Instead, depending on the material, different overtones are played along with the basenote.



Time-domain plot of note A4 bowed on a cello

7.3.1.2 Harmony

Instruments nowadays are tuned based on a, at 440 Hz. An octave higher would be a' at 880 Hz. There are twelve steps from a to a', but their frequencies are not linearly, but exponentially spaced:

- linear: $f_x = \frac{12+x}{12} f_a$
- exponential: $f_x = 2^{\frac{x}{12}} f_a$

Some combinations sound nice together:

- Dur: 0,4,7; $w(t) = w_a(t) + w_{cis}(t) + w_e(t)$
- Dur-Sept: 0,4,7,11; $w(t) = w_a(t) + w_{cis}(t) + w_e(t) + w_{gis}(t)$
- Moll: 0,3,7; $w(t) = w_a(t) + w_c(t) + w_e(t)$
- Moll-Sept:

They do sound nice together when the wave of their sum does display a nice pattern of peaks - that is, when the sum, too, displays a repeating pattern.

7.3.1.3 Songs

Several tones can be played together, thus adding up their waves: $w(t) = \sum_x w_x(t)$ Also, with every beat, new notes may be played. Thus for every beat a new analysis is required.

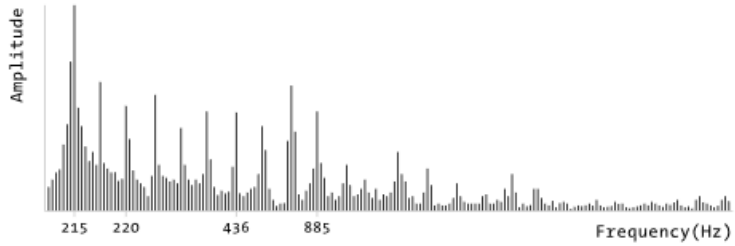
7.3.2 Deconstructing w(t) into fx and ax: the FFT

The sound we hear in a song is really just a timeseries. Its values are the sum of the current amplitudes of all notes, i.o.w. it's just one single, complicated wave $w(t)$. The FFT takes that wave, chops it into windows of, say, a second, and tries to deconstruct the signal in each window into the original, simple waves $w_x(t)$ of the single notes x.

So there are four steps to displaying a songs tones:

1. Chop the signal into parts using the window-function (usually just a step-function). The window-function's length should be equal to the length of one beat.
2. Take the first part and artificially elongate it by duplicating it over and over, thus creating a periodic signal.

3. Do the FFT: represent this periodic signal as a sum of sines of different frequencies and amplitudes.
4. Plot this.
5. Repeat from step two with the next part of the choped-up signal.



Frequency-domain plot of a sustained note (220 Hz) bowed on a cello

7.4 Drawing a room onto a cylinder

The idea here is to imagine a scene taking place in a room. That scene is projected and drawn onto a paper cylinder. This way, a visitor can step inside the cylinder and see the scene around him as if he was actually there.

This requires two projections to happen to the objects in the scene. Fist we need to project them onto the cylinder. Second we need to roll out the cylinder so that we can actually print the scene on it with a printer or by hand.

7.4.1 Projecting a line in space onto a cylinder

We want to project the line from $\vec{a} = (10, 10, 1)$ to $\vec{b} = (0, 10, 1)$ onto a cylinder of radius $r = 1$. We assume that the visitors eyes will be at the coordinates $\vec{c} = (0, 0, 1.6)$.

7.4.1.1 Projecting a point onto the cylinder

7.4.1.2 Projecting a plane onto the cylinder

So we found out where the two edge points of the line should go onto the cylinder. But what would the line connecting them look like? What we need here is the intersection of the cylinder with the plane defined by the points $\vec{a}, \vec{b}, \vec{c}$. That plane is expressed as $P = \{\vec{x} | 0.07y + z = 1.7\}$.

To find the line we need an expression for $P \cap C$, that is, a sollution to the system:

$$0.07y + z = 1.7$$
$$x^2 + y^2 = 1$$

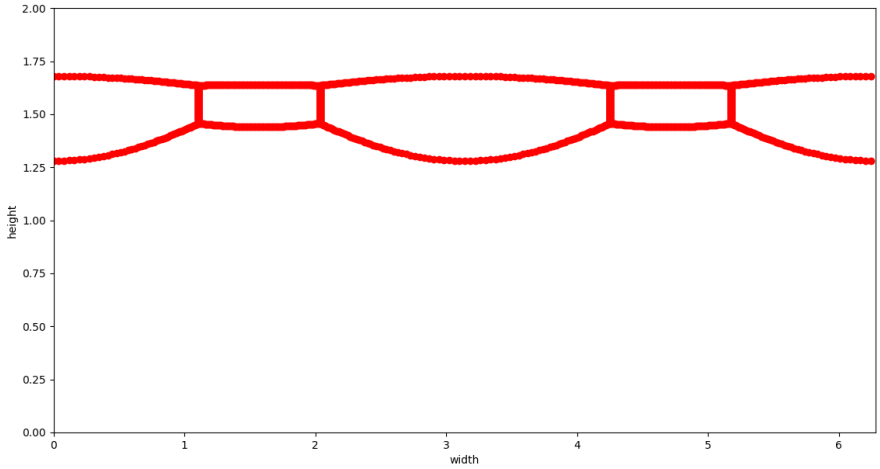
We can factor out y, leaving us with $x^2 + (\frac{1.7-z}{0.07})^2 = 1$

7.4.2 Rolling out the cylinder

We cut the cylinder open along the z axis. That means that for every z, we will apply a mapping $(x, y, z) \to (\theta, z)$. One particular mapping really stands out for that purpose - the polar representation of coordinates. Remember that

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

This means that we can achieve our mapping by using $(x, y, z) \to (cos^{-1}x, z)$.




```
import matplotlib.pyplot as plt
from math import sqrt, pi, acos, atan
```

```
def distance(point1, point2):
    return sqrt((point2[0] - point1[0])**2 +
                (point2[1] - point1[1])**2 +
                (point2[2] - point1[2])**2)


def makeLine(point1, point2, num=100):
    line = []
    deltax = float(point2[0] - point1[0])/num
    deltay = float(point2[1] - point1[1])/num
    deltaz = float(point2[2] - point1[2])/num
    for i in range(num):
        x = point1[0] + i*deltax
        y = point1[1] + i*deltay
        z = point1[2] + i*deltaz
        line.append([x, y, z])
    return line


def intersection(point, head, cylinder):
    rc = cylinder.radius
    xp = point[0]
    yp = point[1]
    xh = head[0]
    yh = head[1]
    alpha1 = (xh**2 - xh*xp + yh**2 - yh*yp - sqrt(rc**2*xh**2 - 2*rc**2*xh*xp + rc**2*xp**2 + rc**2*yh**2 - 2*rc**2*yh*yp
    alpha2 = (xh**2 - xh*xp + yh**2 - yh*yp + sqrt(rc**2*xh**2 - 2*rc**2*xh*xp + rc**2*xp**2 + rc**2*yh**2 - 2*rc**2*yh*yp
    ints1 = [0, 0, 0]
    ints1[0] = head[0] + alpha1*(point[0] - head[0])
    ints1[1] = head[1] + alpha1*(point[1] - head[1])
    ints1[2] = head[2] + alpha1*(point[2] - head[2])
    dist1 = distance(ints1, head)
    ints2 = [0, 0, 0]
    ints2[0] = head[0] + alpha2*(point[0] - head[0])
    ints2[1] = head[1] + alpha2*(point[1] - head[1])
    ints2[2] = head[2] + alpha2*(point[2] - head[2])
    dist2 = distance(ints2, head)
    if dist2 > dist1:
        return ints1
    else:
        return ints2


def toPolar(point):
    pointPolar = [0, 0, 0]
    radius = sqrt(point[0]**2 + point[1]**2)
    #theta = acos(point[0]/radius) * 360.0 / (2*pi)
    theta = atan(point[1]/(point[0] + 0.000001)) * 360.0 / (2*pi)
    if point[0] >= 0 and point[1] >= 0:  # first quad
        theta = theta
    elif point[0] < 0 and point[1] >= 0:  # second quad
        theta = theta + 180
    elif point[0] < 0 and point[1] < 0:  # third quad
        theta = theta + 180
    else:
        theta = theta + 360
    pointPolar[0] = theta
    pointPolar[1] = radius
    pointPolar[2] = point[2]
    return pointPolar


def rollout(point, cylinder):
    pointPolar = toPolar(point)
    newpoint = [0, 0]
    newpoint[0] = 2*pi*cylinder.radius*pointPolar[0]/360
    newpoint[1] = point[2]
    return newpoint


class Cylinder:
    def __init__(self, radius):
        self.radius = radius


cylinder = Cylinder(1)
circumference = 2*pi*cylinder.radius
head = [0, 0, 1.6]
n = 100

# floor
line1 = makeLine([5, 10, 0], [-5, 10, 0], n)
line2 = makeLine([-5, 10, 0], [-5, -10, 0], n)
line3 = makeLine([-5, -10, 0], [5, -10, 0], n)
```

```
line4 = makeLine([5, -10, 0], [5, 10, 0], n)

# walls
line5 = makeLine([5, 10, 0], [5, 10, 2], n)
line6 = makeLine([-5, 10, 0], [-5, 10, 2], n)
line7 = makeLine([-5, -10, 0], [-5, -10, 2], n)
line8 = makeLine([5, -10, 0], [5, -10, 2], n)

# ceiling
line9 = makeLine([5, 10, 2], [-5, 10, 2], n)
line10 = makeLine([-5, 10, 2], [-5, -10, 2], n)
line11 = makeLine([-5, -10, 2], [5, -10, 2], n)
line12 = makeLine([5, -10, 2], [5, 10, 2], n)

line = line1 + line2 + line3 + line4 + line5 + line6 + line7 + line8 + line9 + line10 + line11 + line12


out = []
for i in range(len(line)):
    point = line[i]
    ints = intersection(point, head, cylinder)
    projp = rollout(ints, cylinder)
    out.append(projp)
    pointPolar = toPolar(ints)
    #print "mapping [{},{},{}] via ints [{},{},{}] and via polar [{},{},{}] to flat [{},{}]".format(point[0], point[1], p


xflat = []
yflat = []
for i in range(len(line)):
    xflat.append(out[i][0])
    yflat.append(out[i][1])


plt.plot(xflat, yflat, 'ro')
plt.xlabel('width')
plt.ylabel('height')
plt.axis([0, circumference, 0, 2])
plt.show()
```

7.5 Playing with projections and transformations

I like to start with a simple basefunction, and then distort it.

Nice basefunctions:

- simple geometric
- recursive

Nice distortions:

- sinusoidal (Fourier: add overtones)
- imaginary (Laplace: add more dimensions)
- stepfunction (Mandelbrot: cut off some of them)
- flat onto spherical (Mapprojection)
- circle rolled around circle (Spirograph)
- Copy and rotate
- shadows

Finally, we can refine the result.

- change paras over time
- heatmap
- path

We will try an example now. Take a simple geometric shape. Create it's fourier representation. Add some overtones to it. Convert it back into geometric space. What do you get?

There are basically two ways we can achive this. The first is using ordinary (that is, one-dimenstional) Fourier and parametrised shapes:

- a1: Take some parameterised Shape in $\mathbb{R}^3$: $S(f(t))$
- a2: Extract $f(t)$
- a3: Express $f(t)$ in its Fourier representation: $f(t) = \sum \alpha_i b_i$, where b_i is one-dimensional
- a4: add some overtones to obtain $f'(t)$
- a5: Obtain S' as $S(f'(t))$

The second approach seems simpler, but requires tree dimensional Fourier:

- b1 (= a2): Take a shape $S(\vec{v})$
- b2 (= a3): $S(\vec{v}) = \sum \alpha_i \vec{b_i}$
- b3 (= a4, a5): Obtain S' by adding overtones to S

7.6 Fourier on sound and geometric objects

7.6.1 Basics: Fourier in one dimension

7.6.1.1 Representing a function under a Fourier base

The Fourier functions $\{sin(nt), cos(nt) | n \in \mathbb{N}\}$ form an orthogonal base for all functions in the space of square-integrable functions that are periodic between $-\pi$ and π. Lets first get some familiarity with Fourier analysis.

At first we'll try to express a simple function in a fourier base:

$$f(t) = sin(t) = \alpha_0 + \sum_n^\infty \alpha_n cos(nt) + \sum_n^\infty \beta_n sin(nt)$$

It's comparatively easy to calculate the coefficients of this series[1]:

$$\alpha_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} sin(t)\,dt = 0$$
$$\alpha_1 = \frac{1}{\pi}\int_{-\pi}^{\pi} sin(t)cos(t)\,dt = 0$$
$$\alpha_2 = .. = 0$$
$$\alpha_3 = ... = 0$$

$$\beta_1 = \frac{1}{\pi}\int_{-\pi}^{\pi} sin(t)sin(t)\,dt = 1$$
$$\beta_2 = \frac{1}{\pi}\int_{-\pi}^{\pi} sin(t)sin(2t)\,dt = 0$$
$$\beta_3 = ... = 0$$

In other words: the only amplitude different from 0 is $\beta_1 = 1$

We'll be using python for our Fourier-analysis, where it is more common to work with the exponential/imaginary representation of the Fourier functions. This representation is based on Eulers formula:

$$e^{it} = \cos(t) + i\sin(t)$$

Using this equation we can rewrite the Fourier base as $\{e^{int} | n \in \mathbb{N}\}$. Using this base we get as the only nonzero coefficient:

$$\gamma_1 = \frac{1}{\pi}\int_{-\pi}^{\pi} sin(t)e^{it}\,dt = i$$

Having imaginary amplitudes may be intimidating at first, but really it's very simple. The real part of such an amplitude equals the amplitude of the equivalent cos term, whereas the imaginary part equals the amplitude of the sin term.

7.6.1.2 Bringing in arbitrary intervals

In the previous section, when we wrote the index n, we meant that the function would repeat itself n times within the interval $[-\pi, \pi]$ - it has a frequency of $f_n = \frac{n}{2\pi}$.

However, in reality we will deal with functions that are periodic over an unspecified interval $[-T, T]$ [2]. To acomodate this, we can expand our notion of Fourier basis to the space of all square integrable functions that are periodic between $[-T, T]$. Our base now consists of $\{sin(\frac{2\pi n}{T}t), cos(\frac{2\pi n}{T}t) | n \in \mathbb{N}\}$ or $\{e^{i\frac{2\pi n}{T}t} | n \in \mathbb{N}\}$.

Lets consider the function $f(t) = \sin(\frac{2\pi}{T}t)$. Just like before, it's easy to calculate this functions Fourier coefficients, and just like before, only one of them is nonzero:

$$\gamma_1 = \frac{1}{T}\int_{-T}^{T} \sin(\frac{2\pi}{T}t)e^{i\frac{2\pi}{T}t}\,dt = i$$

The frequency that goes associated with the index $n = 1$ is $f_n = \frac{n}{T} = 0.1$.

[1] The term $\frac{1}{\pi}$ is not strictly neccessary for an orthogonal base, but it is very convenient, since it makes our base ortho*normal*.

[2] In *real* reality, we will not deal with periodic functions at all. But more on that later

7.6.1.3 Implementation and verification in python

It's time to get our hands dirty and learn about python's implementation of Fourier transforms. We'll try to use pythons `fft` library to recreate the previous section:

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

delta = 0.01
T = 10
t = np.arange(-T, T, delta)
data = np.sin(2 * np.pi * t / T)
amps = np.fft.fft(data)
frqs = np.fft.fftfreq(data.size, delta)

f, (ax1, ax2) = plt.subplots(1, 2)
ax1.plot(frqs, np.real(amps))
ax2.plot(frqs, np.imag(amps))
plt.show()
```

As expected, the plot shows a high amplitude at the frequency 0.1. Not quite as expected is the other peak at frequency -0.1. What's up with that? In fact, there are a few phenomena that warrant further explanation:

- Amplitudes are not really 0 where they should be, they only are very close to 0. This is merely an artifact of numerical computation that we won't bother with any further.
- The amplitudes are not normalized to T. This, too, should not pose any difficulties for us, as we will not be using the concrete values of the amplitudes in this project.
- How does fft chose which frequencies to consider? Here, for some reason, it's between -50 and 50.
- Why are there positive and negative frequencies?
- For now, we have chosen `t` to cover the interval $[-T, T]$ perfectly. But what if we have `t` too long or too short? Surprisingly, this doesn't seem to change the estimated frequencies very much.

7.6.1.4 Our first frequency domain operations: adding overtones

We went trough all the trouble of representing $f(t)$ over a Fourier base for a reason: there are operations that are natural in the frequency domain that don't exactly come easily in time domain. One of those operations would be adding overtones.

```
import numpy as np
import matplotlib.pyplot as plt
import simpleaudio as sa


sampleRate = 44100


def addXHalfTonesTo(basefreq, steps):
    return basefreq * (2 ** (steps / 12.0))


def play(data, sampleRate):
    dataNrm = data * 32767 / np.max( np.abs( data ))
    data16 = dataNrm.astype(np.int16)
    return sa.play_buffer(data16, 1, 2, sampleRate)



# creating the data
delta = 1.0 / sampleRate
T = 1
t = np.arange(0, T, delta)
data = np.sin( t * 440 * 2 * np.pi )
#play(data, sampleRate)


# transform to frequency domain
amps = np.fft.fft(data)
frqs = np.fft.fftfreq(data.size, delta)

# filter out the less important frqs
# add chord on top of basetone
thrsh = 0.1 * np.max(np.abs(amps))
ampsNew = np.zeros(np.shape(amps), dtype=np.complex128)
```

```
for i, a in enumerate(amps):
    if np.abs(a) > thrsh:
        f1 = frqs[i]
        f2 = np.round(addXHalfTonesTo(f1, 4))
        f3 = np.round(addXHalfTonesTo(f1, 7))
        i2 = np.where(frqs == f2)[0]
        i3 = np.where(frqs == f3)[0]
        ampsNew[i] += a
        ampsNew[i2] += a
        ampsNew[i3] += a

# convert back to timedomain
dataNew = np.fft.ifft(ampsNew)
play(dataNew, sampleRate)


# plot
f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
ax1.plot(t, data)
ax2.plot(frqs, np.abs(amps))
ax3.plot(t, dataNew)
ax4.plot(frqs, np.abs(ampsNew))
plt.show()
```

7.6.1.5 Windowing

Now, in reality we will not be dealing with a allways constant sinewave. A piece of music is composed of different tones at different times. If we don't account for that, we get something like this:

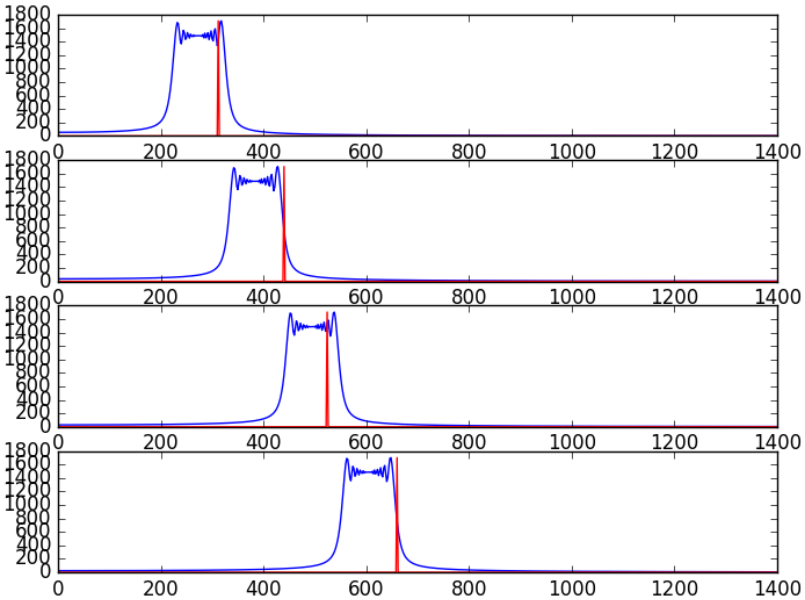
```
import numpy as np
import matplotlib.pyplot as plt
import simpleaudio as sa


sampleRate = 44100


def addXHalfTonesTo(basefreq, steps):
    return basefreq * (2 ** (steps / 12.0))


def play(data, sampleRate):
    dataNrm = data * 32767 / np.max( np.abs( data ))
    data16 = dataNrm.astype(np.int16)
    return sa.play_buffer(data16, 1, 2, sampleRate)



# creating the data
delta = 1.0 / sampleRate
T = 1
t = np.arange(0, T, delta)
fa = 440
fc = addXHalfTonesTo(fa, 4)
fd = addXHalfTonesTo(fc, 3)
a = np.sin( t * fa * 2 * np.pi )
c = np.sin( t * fc * 2 * np.pi )
d = np.sin( t * fd * 2 * np.pi )
t = np.concatenate((t, t, t, t))
data = np.concatenate((a, c, d, c))
#play(data, sampleRate)


# transform to frequency domain
amps = np.fft.fft(data)
frqs = np.fft.fftfreq(data.size, delta)

# filter out the less important frqs
# add chord on top of basetone
thrsh = 0.1 * np.max(np.abs(amps))
ampsNew = np.zeros(np.shape(amps), dtype=np.complex128)
for i, a in enumerate(amps):
    if np.abs(a) > thrsh:
        f1 = frqs[i]
        f2 = np.round(addXHalfTonesTo(f1, 4))
        f3 = np.round(addXHalfTonesTo(f1, 7))
        i2 = np.where(frqs == f2)[0]
        i3 = np.where(frqs == f3)[0]
        ampsNew[i] += a
        ampsNew[i2] += a
        ampsNew[i3] += a
```

```
# convert back to timedomain
dataNew = np.fft.ifft(ampsNew)
play(dataNew, sampleRate)


# plot
f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
ax1.plot(t, data)
ax2.plot(frqs, np.abs(amps))
ax3.plot(t, dataNew)
ax4.plot(frqs, np.abs(ampsNew))
plt.show()
```

Thats why we use windowing. Windowing is the process of cutting an incomming signal in small chunks, for wich we create an individual Fourier analysis. This way, at tone c, we don't have to drag along tone a any more; a has long been forgotten by the analysis.

```
import numpy as np
import matplotlib.pyplot as plt
import simpleaudio as sa


sampleRate = 44100


def first(data, cond):
    for i, d in enumerate(data):
        if cond(d):
            return i, d


def addXHalfTonesTo(basefreq, steps):
    return basefreq * (2 ** (steps / 12.0))


def overtone(steps, frqs, amps):
    ampsNew = np.zeros(np.shape(amps), dtype=np.complex128)
    for i, a in enumerate(amps):
        if np.abs(a) > 0:
            f1 = frqs[i]
            f2 = np.round(addXHalfTonesTo(f1, steps))
            i2 = np.where(frqs == f2)
            #ampsNew[i] += a
            ampsNew[i2] += a
    return ampsNew


def play(data, sampleRate):
    dataNrm = data * 32767 / np.max( np.abs( data ))
    data16 = dataNrm.astype(np.int16)
    po = sa.play_buffer(data16, 1, 2, sampleRate)
    po.wait_done()
    return po


def filterLower(amps, perc):
    thrsh = perc * np.max( np.abs( amps ))
    ampsNew = np.zeros(np.shape(amps), dtype=np.complex128)
    for i, a in enumerate(amps):
        if np.abs(a) > thrsh:
            ampsNew[i] = a
    return ampsNew


def distort(frqs, amps):
    ampsFiltered = filterLower(amps, 0.5)
    terz = overtone(4,frqs, ampsFiltered)
    quint = overtone(7, frqs, ampsFiltered)
    return ampsFiltered + terz + quint


def autotune(frqs, amps):
    ampsFiltered = filterLower(amps, 0.9999)
    ampsTuned = np.zeros( np.shape( ampsFiltered), dtype=np.complex128 )
    halfToneFreqs = [addXHalfTonesTo(220, x) for x in range(23)]
    for i, a in enumerate(ampsFiltered):
        if np.abs(a) > 0:
            f = frqs[i]
            if f > 0:
                ihh, fhh = first(halfToneFreqs, lambda fc: fc >= f)
                fhl = halfToneFreqs[ihh-1]
                disth = np.abs(fhh - f)
                distl = np.abs(fhl - f)
                fh = fhl if distl < disth else fhh
```

```
                inew, fnew = first(frqs, lambda fc: fc >= fh)
                ampsTuned[inew] = a
    return ampsTuned


def chunks(data, length):
    C = int( np.ceil( len(data) / length ))
    chunks = []
    start = 0
    end = length
    for c in range(C):
        chunks.append(data[start:end])
        start += length
        end += length
    return chunks


delta = 1.0 / sampleRate
t = np.arange(0, 2, delta)
freq = lambda t: 220 + 220 * t / 2
data = np.sin(t * freq(t) * 2 * np.pi)

origAmps = []
tunedAmps = []
tunedChunks = []
chunksize = int(data.size / 4)
for chunk in chunks(data, chunksize):
    frqs = np.fft.fftfreq(chunk.size, delta)
    amps = np.fft.fft(chunk)
    ampsNew = autotune(frqs, amps)
    dataNew = np.fft.ifft(ampsNew)
    origAmps.append(amps)
    tunedAmps.append(ampsNew)
    tunedChunks.append(dataNew)


dataNew = np.concatenate(tunedChunks)
play(data, sampleRate)
play(dataNew, sampleRate)


start = 0
stop = 700
f, axes = plt.subplots(4)
for i in range(4):
    axes[i].plot(frqs[start:stop], np.abs(origAmps[i][start:stop]))
    axes[i].plot(frqs[start:stop], np.abs(tunedAmps[i][start:stop]), 'r')
plt.show()
```



7.6.2 Fourier in more than one dimensions

7.6.2.1 Mutliple input parameters

Consider a simple black and white image. We'll describe the value at any pixel as $v(x, y)$, that is, $v : \mathbb{R}^2 \to \mathbb{R}$. Again, the base of the space of $(\mathbb{R}^2 \to \mathbb{R})$ functions must consist of functions that are themselves $(\mathbb{R}^2 \to \mathbb{R})$. The Fourier-representation of an image can be quite naturally obtained by:

$$v(x, y) = \sum_{n}^{\infty} \sum_{m}^{\infty} \alpha_{n,m} b_{n,m}(x, y)$$

where $b_{n,m}(x, y) = e^{i(\frac{n2\pi}{b_x - a_x} x + \frac{m2\pi}{b_y - a_y} y)}$ and

$$\alpha_{n,m} = \int_{a_x}^{b_x} \int_{a_y}^{b_y} v(x, y) e^{i(\frac{n2\pi}{b_x - a_x} x + \frac{m2\pi}{b_y - a_y} y)} \, dy \, dx$$

Generally, when a function takes two parameters, here x, y, then we also need two frequency parameters, here n, m. Analogously, for functions of three parameters we'd need three frequency parameters.

7.6.2.2 Multiple output parameters: Vector valued functions

In the previous section, we dealt with the space of functions mapping $\mathbb{R}^2 \to \mathbb{R}$. How about vector valued functions? Finally, we have arrived at the top level of abstraction when it comes to Fourier representations: the space of functions mapping $\mathbb{R}^a \to \mathbb{R}^b$.

For example, consider the surface $\{\vec{v} | v_z = v_x^2 + 3v_y\}$. The set-expression can be rewritten into a function-expression:

$$\vec{v}(x, y) = \begin{bmatrix} x \\ y \\ x^2 + 3y \end{bmatrix}$$

Here, we have an equation with two parameters (v_x and v_y) mapping onto a 3d-vector $\vec{v}$. Consequently, the Fourier base of the space of surfaces, too, must consist of funcitons mapping $\mathbb{R}^2 \to \mathbb{R}^3$.

As another example, consider an ellipsoid $\{\vec{v} | \frac{x^2}{r_1^2} + \frac{z^2}{r_2^2} + \frac{z^2}{r_3^2} = 1\}$. This body can be parameterized as:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r_1 \cos\theta \cos\phi \\ r_2 \cos\theta \sin\phi \\ r_3 \sin\theta \end{bmatrix}$$

This means we can represent the ellipsoid function as a sum of Fourier-base functions that map $[0°, 360°]^2 \to \mathbb{R}^3$.

In general, we can represent any function mapping $\mathbb{R}^a \to \mathbb{R}^b$ by a Fourier base of functions from the same space. As for geometric bodies: as long as we can find a parameterisation of the body (that is continuous and sqare-integrable), we can also find a Fourier-representation[3].

Surprisingly, this works just the same. Consider the ellipsoid in its parameterized representation:

$$\vec{v}(\phi, \theta) = \begin{bmatrix} r_x \cos\phi \cos\theta \\ r_y \cos\phi \sin\theta \\ r_z \sin\phi \end{bmatrix}$$

$\vec{v}(\phi, \theta)$ is a member of the space $S = \{f : \mathbb{R}^2 \to \mathbb{R}^3\}$, so the basis must consist of functions of the same shape. Let's call one of the members of the basis $\vec{b}_{n,m}$. It must have the following structure:

$$\vec{b}_{n,m}(\phi, \theta) = \begin{bmatrix} b_{n,m}^x(\phi, \theta) \\ b_{n,m}^y(\phi, \theta) \\ b_{n,m}^z(\phi, \theta) \end{bmatrix}$$

Then the coefficient $\alpha_{n,m}$ is calculated like this:

[3] We cannot find a Fourier-representation of non-parameterisable sets. The reason for this is that sets don't form an inner-product space - or at least not without some whacky definitions.

$$\alpha_{n,m} = \int_{a_\theta}^{b_\theta} \int_{a_\phi}^{b_\phi} \vec{v}(\phi,\theta)\vec{b}_{n,m}(\phi,\theta)\, d\phi\, d\theta$$

$$= \int_{a_\theta}^{b_\theta} \int_{a_\phi}^{b_\phi} (\vec{i}v^x(\phi,\theta) + \vec{j}v^y(\phi,\theta)\vec{k}v^z(\phi,\theta))(\vec{i}b^x_{n,m}(\phi,\theta) + \vec{j}b^y_{n,m}(\phi,\theta)\vec{k}b^z_{n,m}(\phi,\theta))\, d\phi\, d\theta$$

$$= \int_{a_\theta}^{b_\theta} \int_{a_\phi}^{b_\phi} 1v^x(\phi,\theta)b^x_{n,m}(\phi,\theta) + 0 + 0 + 0 + 1v^y(\phi,\theta)b^y_{n,m}(\phi,\theta) + 0 + 0 + 0 + v^z(\phi,\theta)b^z_{n,m}(\phi,\theta)\, d\phi\, d\theta$$

$$= \int_{a_\theta}^{b_\theta} \int_{a_\phi}^{b_\phi} v^x(\phi,\theta)b^x_{n,m}(\phi,\theta)\, d\phi\, d\theta + \int_{a_\theta}^{b_\theta} \int_{a_\phi}^{b_\phi} v^y(\phi,\theta)b^y_{n,m}(\phi,\theta)\, d\phi\, d\theta + \int_{a_\theta}^{b_\theta} \int_{a_\phi}^{b_\phi} v^z(\phi,\theta)b^z_{n,m}(\phi,\theta)\, d\phi\, d\theta$$

Let's apply the Fourier base $\begin{bmatrix} b^x_{n,m}(\phi,\theta) \\ b^y_{n,m}(\phi,\theta) \\ b^z_{n,m}(\phi,\theta) \end{bmatrix} = \begin{bmatrix} e^{i(\frac{n2\pi}{b_\phi - a_\phi}\phi + \frac{m2\pi}{b_\theta - a_\theta}\theta)} \\ e^{i(\frac{n2\pi}{b_\phi - a_\phi}\phi + \frac{m2\pi}{b_\theta - a_\theta}\theta)} \\ e^{i(\frac{n2\pi}{b_\phi - a_\phi}\phi + \frac{m2\pi}{b_\theta - a_\theta}\theta)} \end{bmatrix}$ and the ellipsoid formula to this general expression. Then we get:

$$\alpha_{n,m} = \int_{a_\theta}^{b_\theta} \int_{a_\phi}^{b_\phi} r_x \cos\phi\cos\theta e^{i(\frac{n2\pi}{b_\phi - a_\phi}\phi + \frac{m2\pi}{b_\theta - a_\theta}\theta)}\, d\phi\, d\theta + \int_{a_\theta}^{b_\theta} \int_{a_\phi}^{b_\phi} r_y \cos\phi\sin\theta e^{i(\frac{n2\pi}{b_\phi - a_\phi}\phi + \frac{m2\pi}{b_\theta - a_\theta}\theta)}\, d\phi\, d\theta + \int_{a_\theta}^{b_\theta} \int_{a_\phi}^{b_\phi} r_z \sin\phi e^{i(\frac{n2\pi}{b_\phi - a_\phi}\phi}$$

$$\alpha_{n,m} = r_x \int_0^{2\pi} \int_0^{2\pi} \cos\phi\cos\theta e^{i(n\phi + m\theta)}\, d\phi\, d\theta + r_y \int_0^{2\pi} \int_0^{2\pi} \cos\phi\sin\theta e^{i(n\phi + m\theta)}\, d\phi\, d\theta + t_z \int_0^{2\pi} \int_0^{2\pi} \sin\phi e^{i(n\phi + m\theta)}\, d\phi\, d\theta$$

Integrating yields:

...

Of course, the whole point is to graph the thing:

```
import numpy as np

def ellipsoid(theta, phi, rx, ry, rz):
    x = rx * np.cos(theta) * np.cos(phi)
    y = ry * np.cos(theta) * np.sin(phi)
    z = rz * np.sin(theta)
    return x, y, z


def filterAmps(data, perc=0.8):
    dataAbs = np.abs(data)
    x, y = np.shape(data)
    thrsh = perc * np.max(dataAbs)
    filtered = np.zeros((x,y), dtype=np.complex128)
    for c in range(x):
        for r in range(y):
            if dataAbs[c,r] >= thrsh:
                filtered[c,r] = data[c,r]
    return filtered


def addOvertones(amps):
    x, y = np.shape(amps)
    ampsNew = np.zeros( (x, y), dtype=np.complex128 )
    for c in range(x):
        for r in range(y):
            if np.abs(amps[c,r]) > 0:
                c2 = 2*c if 2*c < x else int(c/2)
                r2 = 2*r if 2*r < y else int(r/2)
                ampsNew[c,r] += amps[c,r]
                ampsNew[c,r2] += amps[c,r]
                ampsNew[c2,r] += amps[c,r]
                ampsNew[c2,r2] += amps[c,r]
    return ampsNew


rx = ry = rz = 1
```

```
data = np.zeros((360, 360, 3), dtype=np.float)
for t in np.arange(0,360):
    for p in np.arange(0, 360):
        data[t, p] = ellipsoid(t, p, rx, ry, rz)


ampsx = np.fft.rfft2(data[:,:,0])
ampsy = np.fft.rfft2(data[:,:,1])
ampsz = np.fft.rfft2(data[:,:,2])

ampsxF = filterAmps(ampsx, 0.001)
ampsyF = filterAmps(ampsy, 0.001)
ampszF = filterAmps(ampsz, 0.001)
ampsxF = addOvertones(ampsxF)
ampsyF = addOvertones(ampsyF)
ampszF = addOvertones(ampszF)

dataNew = np.zeros((360,360,3), dtype=np.float)
dataNew[:,:,0] = np.fft.irfft2(ampsxF)
dataNew[:,:,1] = np.fft.irfft2(ampsyF)
dataNew[:,:,2] = np.fft.irfft2(ampszF)
```

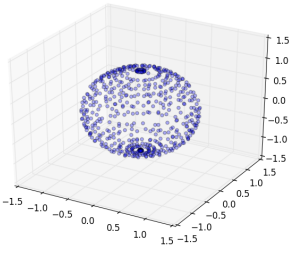


Figure 7.1: Original ellipsoid

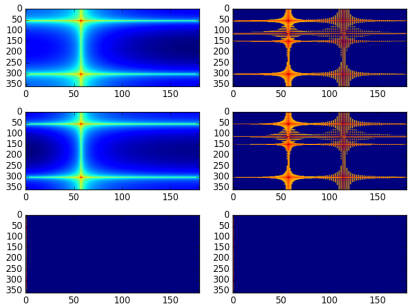


Figure 7.2: Left: original spectra, right: adjusted spectra

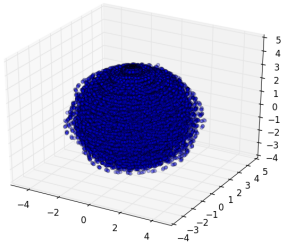


Figure 7.3: Ellipsoid backtransformed

7.7 Estimating the time needed for a task

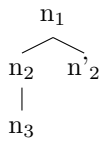
Lets model the growth of a tree like this: we begin with the trunk. From the trunk, branches may grow. How many branches is decided by a Poisson-distribution, whose expected number of branches is dependent on the height from the ground.

```
from scipy.stats import poisson
from collections import namedtuple


Node = namedtuple('Node', 'parent, children')


def lamPerHeight(height):
    return 4 - height * 4 // 7


def randomTree(height=0):
    tree = Node(parent=None, children=[])
    lam = lamPerHeight(height)
    nrChildren = poisson.rvs(lam, size=1)
    for i in range(nrChildren):
        child = randomTree(height + 1)
        tree.children.append(child)
    return tree

print randomTree()
```

Using a Poisson-distribution, the probability of there being k branches on height h would be

$$\underset{h}{pois}(k) = \frac{e^{-\lambda_h}\lambda_h^k}{k!}$$

From the probability of a single branch we can now model the probabilty of a *tree*. Consider the following *tree*:

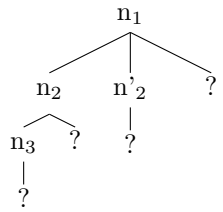


For this *tree*the probability is $P(tree) = pois_1(2)\, pois_2(1)\, pois_3(0)\, pois_2(0)$, or more generally:

$$P(tree) = \underset{1}{pois}(|\, children\,|) \prod_{children} P(childtree)$$

```
def prob(tree, height=0):
    p = 1
    nrChildren = len(tree.children)
    lam = lamPerHeight(height)
    p *= poisson.pmf(nrChildren, lam)
    for child in tree.children:
        p *= prob(child, height + 1)
    return p


t = randomTree()
pt = prob(t)
```

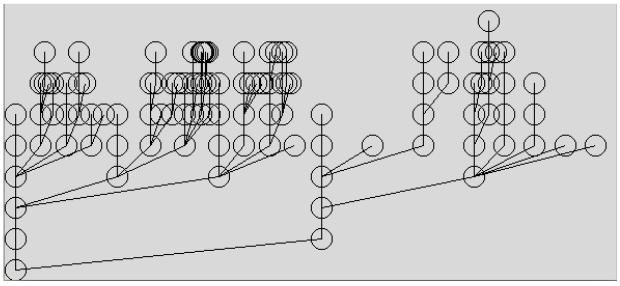
Now what about this tree:



This is an unfinished tree. It's probability is $P(\{tree\,|\,tree_0 \subseteq tree\}) = pois_1(\geq 2)\, pois_2(\geq 1)\, pois_3(\geq 0)\, pois_2(\geq 0) = pois_1(\geq 2)\, pois_2(\geq 1)$, or more generally:

$$P(\{tree\,|\,\underset{0}{tree} \subseteq tree\}) = \underset{1}{pois}(\geq |\, children\,|) \prod_{children} P(\{childtree|childtree_0 \subseteq childtree\})$$

Figure 7.4: Using this model, we can spawn random trees



```python
def probUnfinished(tree, height=0):
    p = 1
    nrChildren = len(tree.children)
    lam = lamPerHeight(height)
    p *= (1 - poisson.cdf(nrChildren, lam))
    for child in tree.children:
        p *= probUnfinished(child, height + 1)
    return p
```

Time as random variable Let time T be a random variable of a tree *tree* defined as:

$$T_b(tree) = T_n(base) + \sum T_b(child)$$

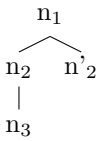
The expexted time for a not yet grown tree would then be:

$$E[T_b] = \sum_{tree \in \Omega} T_b(tree) P(tree)$$

And the expected time given that we already have the fist few nodes of a tree:

$$E[T_b]_{|\, tree_0} = \sum_{tree \in \Omega} T_b(tree) P(tree \,|\, \underset{0}{tree})$$

This sum is pretty hard to compute, since Ω is an infinite set. But there is a way around this problem. Consider this tree:



What is the expected number k of nodes on level 2, given that there are already two offspring?

$$E[k]|k \geq 2 = \sum_{k=0}^{\infty} k P(k|2) = \sum_{k=2}^{\infty} k \frac{pois_{\lambda_2}(k)}{pois_{\lambda_2}(\geq 2)}$$

$$= \frac{1}{pois_{\lambda_2}(\geq 2)} [\sum_{k=0}^{\infty} k \underset{\lambda_2}{pois}(k) - \sum_{k=0}^{1} k \underset{\lambda_2}{pois}(k)]$$

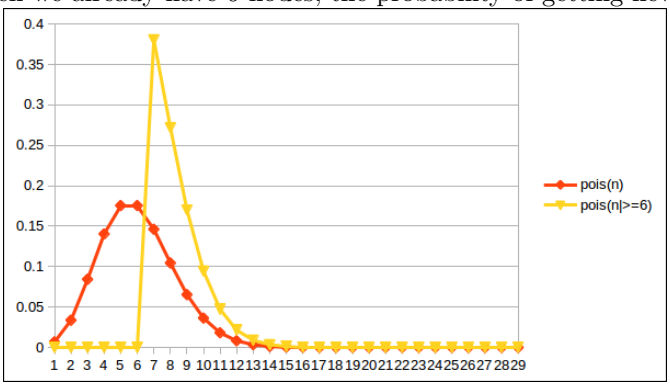
$$= \frac{1}{pois_{\lambda_2}(\geq 2)} [E[k] - \sum_{k=0}^{1} k \underset{\lambda_2}{pois}(k)]$$

$$= \frac{\lambda - \sum_{k=0}^{k=1} k \, pois_{\lambda_2}(k)}{1 - \sum_{k=0}^{k=1} pois_{\lambda_2}(k)}$$

$$= \frac{\lambda - e^{-\lambda} \sum_{k=0}^{k=1} k \lambda^k / k!}{1 - e^{-\lambda} \sum_{k=0}^{k=1} \lambda^k / k!}$$

We can check our predictions numerically:

Figure 7.5: When we already have 6 nodes, the probability of getting new nodes changes




```python
from math import factorial, exp, floor


def pois(lmbd, k):
    return exp(-lmbd) * lmbd**k / factorial(k)


def poisCuml(lmbd, k):
    p = 0
    for j in range(k+1):
        p += pois(lmbd, j)
    return p


def poisCond(lmbd, k, k0):
    pb = 1 - poisCuml(lmbd, k0 - 1)
    if k < k0:
        pa = 0
    else:
        pa = pois(lmbd, k)
    return pa / pb


def expPois(lmbd):
    s = 0
    for i in range(int(50*lmbd)):
        s += i * pois(lmbd, i)
    return s


def expPoisCond(lmbd, k0):
    e = 0
    for i in range(k0, int(50*lmbd)):
        e += i * poisCond(lmbd, i, k0)
    return e


def expPoisCondAnal(lmbd, k0):
    sum1 = sum2 = 0
    for k in range(k0 + 1):
        sum1 += k * lmbd**k / float(factorial(k))
    for k in range(k0 + 1):
        sum2 += lmbd**k / float(factorial(k))
    sum1 *= exp(-lmbd)
    sum2 *= exp(-lmbd)
    a = float(lmbd - sum1)
    b = float(1 - sum2)
    return a / b
```

We will make use of this result shortly.

We can use this to find an intuitive expression for $E[T_b(tree)]$, which would be:

$$E[T_b(tree)]_{|\,tree_0} = E[T_n(base) + \sum T_b(child)]_{|\,tree_0}$$

$$= E[T_n(base)]_{|\,tree_0} + E[\sum T_b(child)]_{|\,tree_0}$$

The term $E[\sum T_b(child)]_{|\,tree_0}$ is somewhat special: we cannot just equal it to $\sum E[T_b(child)]_{|\,tree_0}$, because we don't yet know how many children the tree will have. But it is reasonable to assume that:

$$E[\sum T_b(child)]_{|\,tree_0} = \sum_{children_0} E[T_b(child)] + \sum_{E[|\,children\,|]_{children_0} - children_0} E[T_b(child)]$$

With this, the previous equation becomes:

$$E[T_b(tree)]_{|\,tree_0} = E[T_n(base)]_{|\,tree_0} + \sum_{children_0} E[T_b(child)] + \sum_{E[|\,children\,|]_{children_0} - children_0} E[T_b(child)]$$

Now there are a bunch of terms in this equation that we can approximate statistically:

- $E[|\,children\,|]_{|\,children_0} = \frac{\lambda - e^{-\lambda} \sum_{k=0}^{k=children_0\,-1} k\lambda^k/k!}{1 - e^{-\lambda} \sum_{k=0}^{k=children_0\,-1} \lambda^k/k!}$. Here we can approximate λ by the average number of children on this level.
- $E[T_n(base)]_{|\,tree_0}$ can be approximated by the average net-time on this level
- $E[T_b(child)]$ knows two cases:
  - if we're dealing with a real child, the value is approximated with the same formula applied recursively
  - if we're dealing with a not-yet-spawned child, we have: $E[T_b] = E[T_n(1)] + E[\#c1]E[T_n(2)] + E[\#c1]E[\#c2]E[T_n(3)] + E[\#c1]E[\#c2]E[\#n3]E[T_n(4)] + ...$

Time as random variable: revisited We can enhance our model quite a bit by recognizing that $T_b(base)$ is not a constant, but a random variable in its own right.

Then this random variable would still be defined as:

$$T_b(tree) = T_n(base) + \sum T_b(child)$$

But $T_n(base)$ itself would be random as well.

Under these assumptions, we get:

$$P(t|t_0) = \begin{cases} \frac{P(t)}{1 - P(t<t_0)} & \text{for } t \geq t_0 \\ 0 & \text{for } t < t_0 \end{cases}$$

$$E[T_n(base)]_{|t_0} = \int_0^\infty tP(t|t \geq t_0)\,dt$$

$$= \frac{1}{1 - P(t|t < t_0)} \int_{t_0}^\infty tP(t)\,dt$$

$$= \frac{1}{1 - P(t|t < t_0)} [E[t] - \int_0^{t_0} tP(t)\,dt]$$

Assuming that $P(t) = Gamma_{\alpha,\beta}(t) = \frac{\beta^\alpha e^{-\beta} t^{\alpha-1}}{\Gamma(\alpha)}$, we obtain:

$$\int_0^{t_0} tP(t)\,dt = \frac{\beta^\alpha e^{-\beta}}{\Gamma(\alpha)} \int_0^{t_0} t^{\alpha-1} t\,dt$$

$$= \frac{\beta^\alpha e^{-\beta} t_0^{\alpha-1}}{\Gamma(\alpha)} \frac{t_0^2}{\alpha+1} = \underset{\alpha,\beta}{Gamma}(t_0) \frac{t_0^2}{\alpha+1}$$

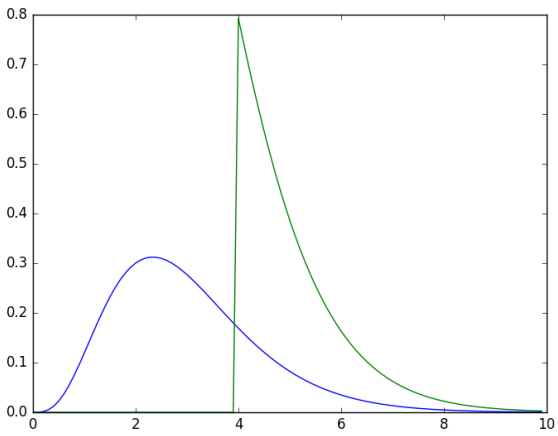
Here, the parameters α and β can be estimated by:

$$\alpha \approx \frac{mean(t)\,mean(t)}{var(t)}$$

$$\beta \approx \frac{mean(t)}{var(t)}$$

We can once again verify this model numerically:

```
import scipy as sp
from scipy.stats import gamma
import matplotlib.pyplot as plt


def rateToScale(beta):
    return 1/beta

def scaleToRate(theta):
    return 1/theta

def probGamma(t, alpha, beta):
    theta = rateToScale(beta)
    return gamma.pdf(t, a=alpha, scale=theta)

def probGammaCond(t, t0, alpha, beta):
    theta = rateToScale(beta)
    gam = gamma(a=alpha, scale=theta)
    tpart1 = t[sp.where(t<t0)]
    outpart1 = sp.zeros(sp.shape(tpart1))
    tpart2 = t[sp.where(t>=t0)]
    pt = gam.pdf(tpart2)
    Pt0 = gam.cdf(t0)
    outpart2 = pt / (1 - Pt0 )
    return sp.concatenate([outpart1, outpart2])

def expGammaCond(t0, alpha, beta):
    theta = rateToScale(beta)
    gam = gamma(a=alpha, scale=theta)
    expv = gam.mean()
    gt0 = gam.pdf(t0)
    gfac = t0 * t0 / (alpha + 1)
    gCumt0 = gam.cdf(t0)
    return ( expv - gt0 * gfac ) / ( 1 - gCumt0 )

mean = 3.0
var = 2.0
alpha = mean * mean / var
beta = mean / var
t0  = 4
t = sp.arange(0, 10, 0.1)
pn = probGamma(t, alpha, beta)
pc = probGammaCond(t, t0, alpha, beta)


plt.plot(t, pn)
plt.plot(t, pc)
plt.show()
```



This model justifies another look at our samplespace. Note that ....

7.8 Scheduling

The proposed algorithm is simple:

1. Take the most urgent remaining task T_i.
2. Put it in the slot $[t_0, t_0 + expt(T_i)]$
3. If $t_0 + expt(T_i) > deadline(T_i)$, throw an exception
4. Set $t_0 = t_0 + expt(T_i)$ and $i = i + 1$ and loop to step 1.

Doublecheck: What if the task has scheduled subtasks?

7.8.1 Running time

7.8.2 Partial correctness

7.8.3 Termination

7.9 Webcamgiant

7.9.1 Webcam to RPy

7.9.2 RPy to computer

We need some way to transfer the images from the raspberry to our computer. Obviously, my favourite way would be over the internet - but to use the internet, you need a cable (ISDN or DSL) that is connected to some public router. Since we don't have that out in the field, we need to fall back onto a wireless transfer - GSM.

7.9.3 Computer to Oculus

7.10 Labeling radar-movies with likely summer-thunderstorms

Summer-thunderstorms are notoriously hard to predict. But we have an abundance of radar-data, together with decent records of actual precipitation.

- Create labeled data
  - When there is a thunderstorm about to come in the next three hours, label as "upcomming"
  - When there is a storm happening, label as "active"
  - When the storm is receding, label as "receding"
  - Otherwise, label as "clear"
- Train a net to recognize the labels given a 10-hour radar-movie
  - Lower layers: 3d-conv
  - Higher layers: dense
- FInd out what the model has learned
  - Are the conv-patterns interesting? Can they be used to find the outlines of a storm-cell?

7.11 Backpropagation for hydrological parameter estimation

Hydrological models consist of a network - a DAG - of landsurfaces, connected with flow paths. Each landsurface has one parameter c (the flow-constant). It seems like to find optimal values for the c's, we could exploit the network-structure of a catchment and apply the backpropagation algorithm.

- Just like neural nets, we want to minimize an error function.
- Just like neural nets, we can do this by backpropagating the differential of a parameter.
- Unlike neural nets, we don't consider all possible connections between nodes - the elevation-map can take that burden from us.
- Unlike neural nets, each node has state. This might complicate things - but in reality, when the net runs long enough, different initial states converge.

7.11.1 Hydrological model

```
class Node:
    def __init__(self, id, V0, c, a):
        self.id = id
        self.V = V0
        self.c = c
        self.a = a

    def eval(N, qIns):
        inpt = N*self.a + sum(qIns)
        outpt = self.c * self.V
        self.V += (inpt - outpt)
        return outpt


class Model:
    def __init__(self, nodes, connections):
        self.nodes = nodes              #{id: node}
        self.connections = connections  #[(idFrom, idTo)]

    def eval(Ns):    #Ns: {id: n}
        lastNode = self.nodes[0]
        qOut = evalNode(lastNode, Ns)
        return qOut

    def evalNode(self, node, Ns):
        upstreamNeighbours = self.getUpstreamNeighbours(node)
        qs = []
        for neighbour in upstreamNeighbours:
            qUpstream = self.evalNode(neighbour, Ns)
            qs.append(qUpstream)
        qOut = node.eval(Ns[node.id], qs)
        return qOut

    def getUpstreamNeighbours(self, node):
        neighbours = []
        for connection in self.connections:
            if connection[1] == node.id:
                neighbourId = connection[0]
                neighbour = self.nodes[neighbourId]
                neighbours.append(neighbour)
        return neighbours

    def gradients(self, sse):
        pass

    def updateParas(self, gradients):
        pass



def modelFactory(terrainMap):
    nodes = None
    connections = None
    model = Model(nodes, connections)
    return model

def optimize(model, N_timeSeries, q_timeSeries):
    errors = []
    for t, Ns in enumerate(N_timeSeries):
```

```
        qPred = model.eval(Ns)
        qObs = q_timeSeries[t]
        errors.append([qPred - qObs]**2)
    sse = sum(errors)
    grads = model.gradients(sse)
    model.update(grads)



terrainMap = None
N_timeSeries = None
q_timeSeries = None
model = modelFactory(terrainMap)
optimize(model, N_timeSeries, q_timeSeries)
```

7.11.2 Optimisation using tensorflow

7.12 Hydrological models as Hidden Markov Models

Whereas the above section dealt primarily with parameter estimation, here we are going to handle state estimation.