UNIVERSITY OF TWENTE.

Pearls of Computer Science

Module 1, Study *Technical Computer Science*
Academic year 2018/2019

DATE
September 10, 2018

MODULE COORDINATOR
Doina Bucur

DESIGN TEAM
Maurice van Keulen
Arend Rensink
Pieter-Tjerk de Boer

TEACHERS
Pieter-Tjerk de Boer (Pearls 000 and 101)
Arend Rensink (Pearl 001)
Maurice van Keulen (Pearl 010)
Marco Gerards (Pearl 011)
Andreas Peter (Pearl 100)
Gwenn Englebienne, Christin Seifert (Pearl 110)
Klaas Sikkel (Pearl 111)

Contents

Chapter 0

Introduction

This is the introductory chapter of the module guide for "Pearls of Computer Science", the first module of the study *Technical Computer Science*.

0.1 Global overview of the module

This section contains a global overview of the entire module. All parts are described briefly.

The module is organized as 8 "*pearl weeks*" and a final project (see Figure 1). Each pearl focuses on one of the achievements in Computer Science. The main task of each week is the 'pearl assignment', done in pairs of students, for which all knowledge and skills are taught in the one week. Each pearl is assessed with this assignment, as well as a written test. For clarity, the term *assignment* is consistently used to refer to the pearl assignment, and the term *project* to refer to the final project.

The pearls are numbered in binary and have their associated RGB colors:

Pearl 000 *Black pearl — Computer Architecture*
Pearl 001 *Blue pearl — Algorithmics*
Pearl 010 *Green pearl — Databases*
Pearl 011 *Cyan pearl — Functional Programming*
Pearl 100 *Red pearl — Cryptography*
Pearl 101 *Magenta pearl — Computer Networks and Operating Systems*
Pearl 110 *Yellow pearl — Intelligent Interaction*
Pearl 111 *White pearl — Requirements Engineering*

The final project consists of the development (in larger groups of 6 students) of either a Twitter dashboard for a live and massive event such as the 'Batavierenrace'[1] (the default project), or a security solution based on entry passes or tags (see Section 0.5) .

The final project starts in week 8, during pearl 111 about Requirements Engineering. The pearl assignment of this pearl is at the same time the first step for your project, namely specifying the requirements and wishes. In week 7 during pearl Intelligent Interaction, a session is organized providing information about the available tools for the project. Week 9 is entirely dedicated to working on the project. The goal is to

[1] The world's largest relay run, a yearly race from Nijmegen to Enschede.

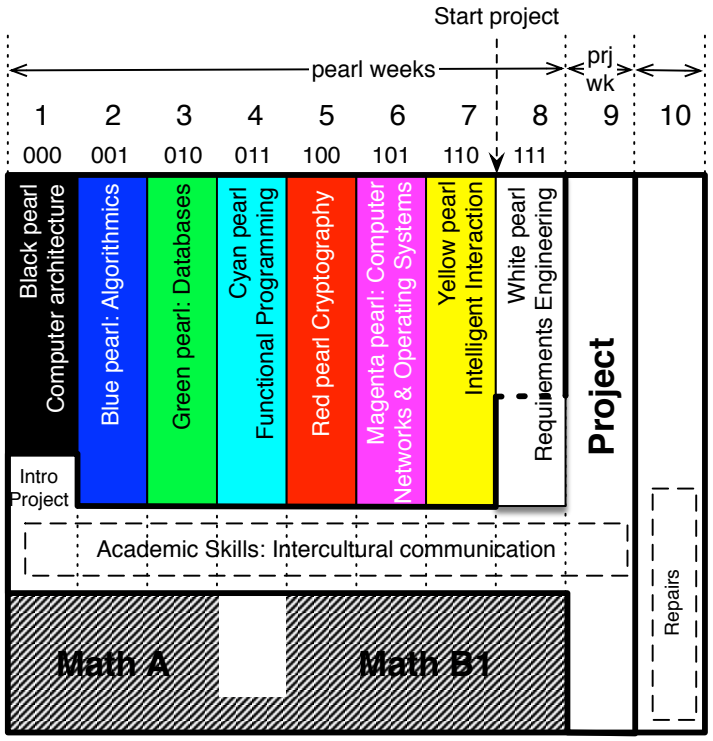


Figure 1: Global overview of the module

finish the project by the end of that week with a presentation on Friday afternoon. Week 10 is completely free unless you have one or more tests or assignments to repair (see Section 0.4.3).

Besides education that immediately pertains to the pearls, you also receive, just like in other modules throughout your study, *math* education and training in *academic skills*. The themes for academic skills in this module are *academic fraud* and becoming an "intercultural communicator". The sessions for academic skills are done in large pre-assigned groups; see Canvas to learn to which group you are assigned and when are where your sessions take place. In Section 0.4.1, you can learn that you receive one grade for the whole module including math. Since math is centrally organized and taught in the same way to many studies in Twente, you need to go to a different Canvas site for information and instructions.

Finally, this module is an integral part of the study Technical Computer Science, but it is also in part followed by students of Business Information Technology students: more concretely, BIT students follow pearl weeks 2, 4, and 6, during which programming and computer networks are studied. Module 2 will be completely shared by TCS and BIT.

0.2 Why (these) pearls?

Computer Science has many sides — probably more than you are aware of at the moment you start your study. The field exists for more than 50 years now; it is not surprising that during that time, much has been invented and developed, on many different topics.

The philosophy behind this module is to give you a small taste of several of these topics at the start of your study. In this way, you can broaden your perspective, you can quickly obtain an idea of the richness and diversity in the study you have chosen, and you can better ascertain which topics you do and do not like.

The topics in this module are called "pearls", because we are proud of them: they pertain to accomplish-

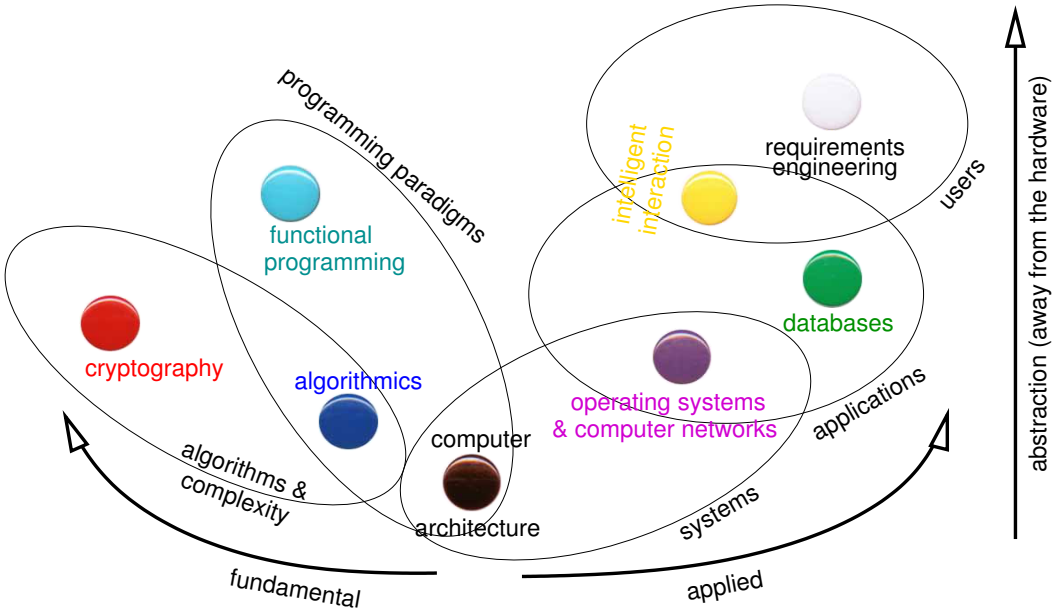


Figure 2: The eight pearls in relation with each other, embedded within larger areas

ments of the field of Computer Science, areas in which fundamental insights and techniques have been developed in this half of a century. We do not, however, claim that these pearls together form a complete and balanced summary of the entire field! Focusing on a few pearls is something completely different than mapping an entire field.

In order to create a global overview, we have positioned the eight pearls in one picture (see Figure 2). Around them we have sketched a few larger areas within which the pearls can be placed. It would go too far and would not be of much use to describe these areas in more detail here. But, by coming back to this "pearl picture" every week, we hope to give you at least some grip.[2]

The two dimensions in Figure 2 roughly represent "from fundamental to applied" (horizontally from left to right) and "from concrete to abstract" (vertically from bottom to top). We move quite arbitrarily through this picture during the quartile.

0.3 Group assignment, presence and deliverables

The practical sessions and pearl assignments are done in pre-assigned pairs students. Each pearl has a different assignment. See Canvas to find out for each pearl who your partner is.

To facilitate that you can easily find your partner, we additionally publish a layout of the lecture room of the first pearl lecture of each week (the one on Monday 6+7 hour) with group numbers in it. If you don't know your partner, go and sit at the indicated spot: you will find your partner there. If you know your partner already or have established contact by some other means, then you can disregard this. You are not obliged to sit at the indicated spot; it is only meant as a service to make it easy for you to find your assignment partner if you don't know them.

You must be present at the tutorial and practical sessions. This is necessary for your grade: the teaching assistants sign-off your assignments. Obviously you can arrange that you can be absent if there is a good reason (for example, because of an illness) and if you inform your assignment partner in advance.

The module guide of each pearl may specify which other sessions are obligatory.

[2] Be aware that Figure 2 is not a generally accepted overview of the field of Computer Science. For the purpose of this module, it suffices.

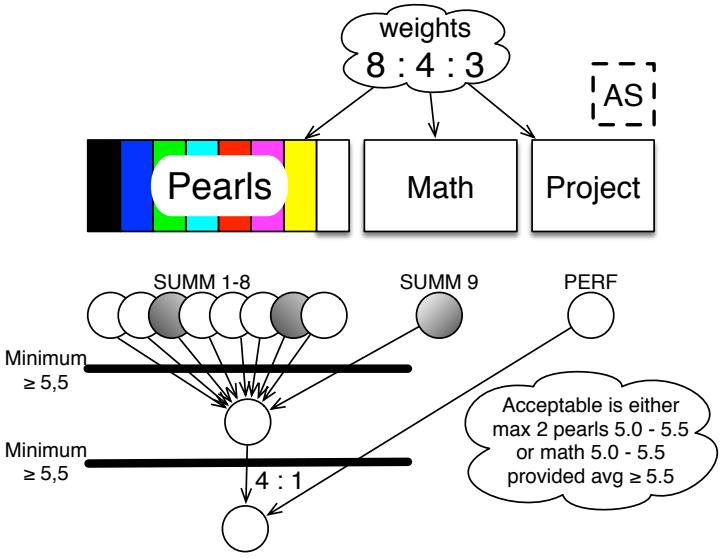


Figure 3: Overview of derivation final grade

Presence at the Math Tutorials is *compulsory* and attendance will be checked. At least 75% of the *supervised* sessions need to be attended; if you attend fewer sessions, you are not allowed to participate in the test.[3]

Your solution(s) to each pearl assignment have to be submitted to Canvas. There is an 'assignment' for each pearl. *Only one* of the assignment partners should submit the solution, but don't forget to write *both your names* on it.

Each pearl has bonus assignments with which you can obtain up to one extra grade point for your pearl test grade. The bonus assignments are meant for the better students to challenge them to go deeper into the subject matter. The *deadline* for the pearl assignments is always Friday night at 23:59. You are allowed, without asking, to extend the deadline to Sunday night 23:59, but as a consequence you lose the right to any bonus.

Not submitting the pearl assignment before Sunday night means no grade for the pearl, hence an insufficient grade for the whole module.

0.4 Assessment and repairs

0.4.1 Assessment

The assessment of the module consists of three components: the pearls, math, and the project (see Figure 3). Academic Skills (AS) is pass/fail: it has no weight in the final grade, but serious participation and a submission of the final assignment are strictly necessary for passing the module.

The three components together determine the final grade for the module by means of a weighted average. The global weights for the three components are 8:4:3, respectively.

Each pearl is assessed with one grade (SUMM1–SUMM8 in Figure 3, where SUMM stands for "summative"). The pearl grade is based on the pearl test and the pearl assignment. The test contains questions about

[3]The purpose of this rule is to help you pass the module. It is our experience in previous years that most students who fail the module, fail it because of mathematics. Moreover, there is a strong correlation between attendance at the tutorials and passing the math exams.

everything that should have been learned, i.e., possibly also about what you should have learned while working on the pearl assignment. You receive no separate grade for the pearl assignment, but completing it sufficiently is obligatory. The pearl assignment can, however, yield the bonus of maximally 1 grade point. The grade for the pearl is equal to the grade of the pearl test plus this bonus. The maximum of the pearl grade, therefore, is 11 points.

The math grade (SUMM9 in Figure 3) is determined by the math teachers based on their tests.

We take a weighted average over all SUMM grades. With a weight of 1 for the pearl grades SUMM1–SUMM8 and a weight of 4 for SUMM9 (the math grade), the mentioned ratio of 8:4 is achieved for the pearls and math, respectively. For this weighted average, a minimum of 5.5 is required.

For each SUMM grade, a minimum of 5.5 is required. A small deviation from this general rule is permitted: Either (a) one or two pearls, or (b) SUMM9 (math), with a grade 5.0–5.5 are allowed, provided that the weighted average is 5.5 or higher (in other words, the grades below 5.5 need to be compensated by higher grades for other pearls or math).

Larger insufficiencies are unacceptable and lead to a failure for the entire module. The tests and pearl assignments can be repaired, however, with certain restrictions (see Section 0.4.3). Other modules have similar 'compensation rules' where partial grades are allowed to be between 5.0 and 5.5 provided some average is above 5.5. Note, however, that the BSA requires that at least 3 of the 4 math components need to be sufficient, so you cannot compensate an insufficient math grade in more than one module.

The project grade (PERF in Figure 3, which stands for "performance") is determined by the project supervisor. This grade should be at least 5.5 as well, which one can obtain if one meets the minimum requirements of the project. The minimum requirements are specified in such a way, that they can be met with a basic level of obtained understanding in combination with sufficient dedication. Nevertheless, the intention is of course that you feel challenged to produce something better than the bare minimum! By using weights 4:1 for SUMM:PERF, one finally obtains the mentioned ratio of 8:4:3 for the pearls, math, and the project, respectively.

0.4.2 Test procedures

The duration of a pearl test is *exactly one hour*. You don't need to remain seated the whole hour; you may leave (quietly!) if you are done more quickly. The official general rule for tests and exams is that you are not allowed to leave within 15 minutes after the start of a test and not within 15 minutes before the end of a test. Students who are officially entitled to extra time (for example, because of dyslexia), need to make themselves known to the supervisor; you are allowed to remain seated for an extra 15 minutes.

It is not allowed to bring any study materials to the test, nor any electronic devices with the exception of a calculator. Individual pearls may impose additional restrictions, for example, on the kind of calculator that is allowed.

It is allowed, however, to bring *one A4 with notes of your own (both sides)*. The A4 may contain text (typed or hand-written) or (scaled) pictures and tables copied from the studie material, other sources, or handmade.

0.4.3 Repairs

It can obviously happen that you do not pass a test, could not complete an assignment in time, or your submission is insufficient, or that you couldn't participate in a test due to an illness. This doesn't immediately mean that you failed the entire module. Within the module, in week 10, a limited range of possibilities for repair are arranged. The following rules apply:

- You are allowed to take part in a re-sit of maximally 3 pearl tests.
- Wednesday afternoon week 10: re-sits of pearl tests 000, 010, 100, and 110, i.e., those of all *odd* weeks (it is possible to do 3 pearl tests in one go).
- Thursday afternoon week 10: re-sits of pearl tests 001, 011, 101, and 111, i.e., those of all *even* weeks (it is possible to do 3 pearl tests in one go).

- Friday afternoon week 10: re-sit of the Math tests.
- If you take part in the re-sit of a pearl test, then the highest grade counts. It is allowed to take part in a re-sit of a pearl test for which you already have a passing grade for the purpose of improving your final grade.
- Insufficient pearl assignments can be repaired as well. You are permitted one attempt at repairing each pearl assignment (i.e., you can repair all eight of them). In the same way, an insufficient project can be repaired. The pearl teacher or project supervisor, respectively, determines the form of this repair, often an additional task.

There are two afternoons reserved for re-sits of the pearl tests. You can repair one or more pearl tests during such an afternoon (3 hours). You will receive all four pearl tests at the start with the intention that you only do those that you want to repair. Students eligible for extra time can remain seated for an extra half an hour.

NB 1: The maximum number of re-sits is 3. If you hand in more than 3 tests, they will all be declared invalid.

NB 2: Make each test on a different sheet of paper. They have to be graded by different teachers!

0.5 The final project

0.5.1 Goal

For the final project, you will develop in groups of 6 students either a Twitter dashboard for a live and massive event such as the "Batavierenrace", or a security system based on electronic passes or tags. Both have to be geared towards the requirements of a self-chosen stakeholder. If you are in doubt which project to choose, our advice is to choose the Twitter dashboard project. The other project is meant for students who are particularly interested in computer architecture and cryptography. This project also demands more deepening of knowledge beyond what has been taught in the respective pearls.

The learning goals of the final project are:

- The student can coherently apply and integrate knowledge and skills in a team and for a project that is based on real-world aspects.
- The student has experienced going through all phases of realizing a software artifact.
- Deepening of knowledge and skills from the pearls
  - **Twitter dashboard**: 010 (Databases), 100 (Cryptography) and 111 (Requirements Engineering).
  - **Security with passes/tags**: 010 (Databases), 000 (Computer Architecture) and 110 (Intelligent Interaction).

Both projects have to be geared to the requirements of a group of stakeholders that you choose yourselves. Examples of stakeholders and their requirements are:

- *Bata Twitter dashboard*
  Stakeholders The Bata organization itself, audience, runners, police, journalists, first aid workers, etc.
  Requirements Stakeholders have to be able to answer their information questions with it: "Where are people having fun?" (audience), "Has someone fallen and where?" (first aid workers), "Did people find it a thrilling and fun race?" (journalists)
- *Security with passes/tags*
  Possible applications Security of buildings, terrains, devices; products that shouldn't leave a shop unpaid; participants of an event receive a tag or bracelet with which they can pay for drinks or gain access to paid parts, etc.
  Stakeholders police, security personnel, participants of events, etc.

Requirements The right persons/products/transactions are or are not allowed; The system should be secured well enough against relevant and costly risks/threats (such as copying a pass/tag, or someone hacking the registration system and then stealing the database).

It is required to base the final project on the given tools (see Section 0.5.2). Also, starting something ambitious and not finishing it, is unacceptable. You have to deliver something that is finished and working, and that has value for the imagined stakeholder.

Twitter dashboard What happens on Twitter can be seen as purely "talking about running", but analysis of tweets can serve many different purposes and stakeholders, for example, organisational bottlenecks, public statisfaction, or support for first aid work. What is said on Twitter can provide information and insight about these purposes. Concrete examples of stakeholders are the Bata organization itself, the audience, the runners, the police, journalists, first aid workers, etc. Examples of information needs are "Where are people having fun?" (audience), "Did someone fall and where?" (first aid workers), "Was the race perceived as thrilling and fun?" (journalists). You will choose yourself which stakeholder and information need(s) you particularly address. The design of the dashboard has to be optimized for this stakeholder and information need(s).

Security with passes/tags Security of terrains, buildings, rooms, vehicles, devices and much more is of great importance for counteracting their misuse. Often, there are different user groups (usually called "roles") who need different kinds of access during specific periods in time or under specific circumstances. The design of a suitable security solution that is well-optimized on a specific situation is not trivial. Usually, it is too expensive of even impossible to secure an object against all possible attacks. Therefore, questions like "which attacks are to be expected" and "which attacks cause the most damage" are important to answer. At the same time, a solution should be robust against practical occurences like the loss of a pass. How easy and how long can someone exploit this? How much effort does it cost to obtain a new pass and what is the damage for an authorized person to have no access for a while? And don't forget that the database of who-has-what-pass and who-has-which-access can be the target of attack: the system needs to be resilient against hackers downloading the database or find out how to manipulate the data remotely.

"Security with passes/tags" can be interpreted broadly. You may also develop a system that ensures that products (with a tag) cannot leave the shop without being paid for. Or a system for which you obtain a bracelet (with a tag) upon entry of a terrain or building (for example, a disco, museum, swimming pool, amusement part, festival) and with which you gain access to those areas you have paid for or with which you can pay for drinks.

The objective of the project is to develop a proof-of-concept, i.e., a demonstration that the intended persons/products/transactions are/are not allowed and that the system is resilient against relevant and costly risks/threats. Software to manage the passes (database with admin interface) has to be developed as well.

Note It can happen that it appears impossible to realize certain security requirements. That is not a problem, provided that you explicitly document against which security threats your system is and is not resilient.

0.5.2 With what tools

Twitter dashboard You receive several tweet collections to choose from, include one with all tweets collected during the Batavierenraces of 2013 and 2014. You need to develop that dashboard in Python using the ECA module. This module has been developed by the UT. It makes it possible to specify rules that act in real-time on incoming tweets and to easily realize a dashboard on the web with graphs and such that can be updated also in real-time. A session organized in Week 7 will explain the concept and how to use it. Realizing a dashboard using it effectively entails only the programming of a set of event-condition-action[4] (ECA) rules that analyse the Twitter data and control the self-designed dashboard. In this way, you can do the project with a minimum of programming experience. On the other hand, everything is just Python for

[4] http://en.wikipedia.org/wiki/Event_condition_action

tweet analysis and HTML/CSS for data visualisation, so if you you want something that is not immediately supported, you may decide as a group to develop the necessary extension(s) yourself within the project.

Besides the tweet collection of the Batavierenrace, several other collections are available, such as other sport events, serious request 2011, certain TV-programs, the weather, etc. In case one tweet collection does not meet your expectations, simply choose another.

Security with passes/tags You should develop the security system, i.e., the proof-of-concept, using the Arduino of Pearl 000 (Computer Architecture). You receive an RFID sensor with which you can detect the presence of a pass and read its ID. Software to manage the passes (the database with admin interface) can be developed in any programming language you want. Obviously you need to make sure that this database, the data on the passes, and what is communicated is properly encrypted if the requirements demand it. You can use the techniques of Pearl 110 (Intelligent Interaction) for this.

0.5.3 Project schedule

The project is introduced in Week 1, but only starts with Pearl 111 (Requirements Engineering) in Week 8 during which the requirements for the system will be specified. This pearl introduced you to requirements engineering, the design cycle and 'formal' project management.

You are allowed to form your own project team. The team has to be formed at the beginning of Week 8. We advise you to already look for potential team members from Week 1. The group composition needs to be registered on Canvas. All groups need to be composed of 6 students. It may happen that the total number of students doesn't allow this. Only in this situation groups of 5 or 7 may be formed.

In Week 7, there is a non-obligatory session where you learn how to work with the available tools.

Week 9 is primarily devoted to the final project. In this week, several sessions with supervision and support are organized. Moreover, there are planned deadlines for delivery of (partial) results, a project evaluation (part of AS), and presentations. Week 9 is composed of two phases: a design and a realization phase with an obligatory delivery deadline and final presentation on Friday.

Week 10 is completely free unless you need to repair one or more tests, assignments, the project or academic skills. The available time in Week 10 is intended for

- preparation for and taking part in re-sits, see Section 0.4.3,
- repairs of assignments that were assessed as insufficient,
- an extra iteration or extension to the project if the project was assessed as insufficient.

0.5.4 Supervision and support

A large team of teaching assistants is available in this module. The teaching assistants (who are more advanced students) are the first you can ask when you have technical questions during the pearl weeks as well as the project week.

Besides that, a project supervisor (a teacher) is assigned to every project team. He/she can give you expert advice, but their main focus it to help you with project management: to make sure that you really have a working and acceptable system on the deadline.

0.6 Learning goals

Pearl 000 *Black pearl — Computer Architecture*

After absorbing the pearl "Computer Architecture"

- The student can work with binary and hexadecimal number representations, binary logic and boolean algebra.
- The student knows the basic architecture of a computer and can work with concepts such as register, memory, address, ALU, clock, program, program counter, instruction and mnemonic.
- The student can write simple programs for a microcomputer in machine language using arithmetic, I/O, and (conditional) jump instructions.

Pearl 001 *Blue pearl — Algorithmics*

After absorbing the pearl "Algorithmics"

- The student can explain the importance of searching and sorting algorithms;
- The student can explain the principle of and differences between linear and binary search methods, as well as between bubble sort and merge sort;
- The student understands the complexity arguments behind the aforementioned algorithms and can analyse which is the best solution in what context;
- The student can apply simple imperative programming concepts: if/then, while, integer variables and arrays;
- The student can program the above algorithms in Python.

Pearl 010 *Green pearl — Databases*

After absorbing the pearl "Databases"

- The student knows the basic concepts of databases
- The student can design a databaseschema for a simple case using ER-modeling.
- The student can realize such a design in a relational DBMS using SQL.
- The student can query and update a relational DBMS with SQL.

Pearl 011 *Cyan pearl — Functional Programming*

After absorbing the pearl "Functional Programming"

- The student can apply basic concepts and syntax of the chosen functional language.
- The student is able to use the concept of function application to define simple functions.
- The student can create simple recursive functions that operate on lists and numbers.
- The student can use list comprehension to solve small problems.
- The student can use induction to prove basic properties of small functions.
- The student is able to express simple algorithms in the chosen functional language.

Pearl 100 *Red pearl — Cryptography*

After absorbing the pearl "Cryptography"

- The student understands symmetric-key encryption: block ciphers and their modes of operation, stream ciphers, and the one-time-pad. He/She knows basic design techniques of such ciphers, such as Feistel networks.
- The student understands asymmetric-key encryption: the RSA cryptosystem and the RSA signature scheme. He/she knows how to use it for key exchange (hybrid encryption).
- The student has learned some necessary background knowledge of elementary number theory (modular arithmetic, Euclidean algorithm) and basic probability theory, for a proper understanding of the above mentioned cryptosystems.

Pearl 101 *Magenta pearl — Computer Networks and Operating Systems*

After absorbing the pearl "Computer Networks and Operating Systems"

- The student can identify and explain the most important tasks of an operating system.
- The student understands the working and layered architecture of packet-switched computer networks, and can reason about the delays occurring in them.
- The student knows the basic working of the internet and internet applications, and protocols like TCP, IP, and HTTP.

Pearl 110 *Yellow pearl — Intelligent Interaction*

After absorbing the pearl "Intelligent Interaction"

- The student knows the basic concepts of artificial intelligence and can design a simple rule-based socially intelligent system.
- The student knows the basic principles of machine learning and can design and execute a classification task with a (black-box) classifier.

Pearl 111 *White pearl — Requirements Engineering*

After absorbing the pearl "Requirements Engineering"

- The student can explain the importance of controlled and predictable realisation of software and project artifacts
- The student knows a few techniques for project management
- The student can derive and formulate requirements as well as acceptance criteria for them
- Besides reaching these learning goals, it is an explicit additional goal of this pearl to formulate requirements and set up a structure for the execution of the project of this module.

Project

After carrying out the project

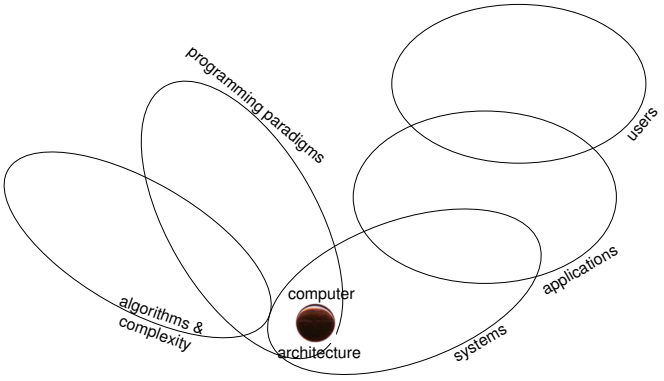
- The student can coherently apply and integrate knowledge and skills in a team and for a project that is based on real-world aspects.
- The student has experienced going through all phases of realizing a software artifact.

Academic Skills

After absorbing and carrying out the exercises of Academic Skills students can use their awareness of their own cultural background and the impact of their background on their intercultural communication to improve their communication with students from different backgrounds. Students will be specifically able to

- The student understands the concepts of fraud and plagiarism, and knows how to behave responsibly as a professional concerning these aspects.
- The student is able to analyse their own culture using the theories described in the book Intercultural Sensitivity.
- The student is able to analyse a culture clash case using the theories described in the book Intercultural Sensitivity.
- The student is able to identify which issues they might encounter working in an intercultural environment and propose ways of dealing with these issues.

The Black Pearl is a real ship
Jack Sparrow — The Curse of the Black Pearl



Week 1

Pearl 000: Computer Architecture

1.1 Introduction

The first pearl of computer science that we will study, is the computer hardware itself. We will focus on two aspects: how can one calculate usefully with only zeros and ones, and how can a machine be built which can be "programmed" to do all kinds of calculations?

1.1.1 Global description of the pearl assignment

This week's pearl assignment is programming a simple small computer (the Arduino), in machine language (we'll see later what that means), such that a LED will flash and a buzzer will make sound. You work on this assignment in groups of two students.

To put things in context: programming in machine language is useful to get a good idea of how a computer works, but, as you will notice, it is quite complicated and error-prone. Therefore, in practice, programs are usually written in a so-called high-level language, and translated automatically (by another computer program!) into machine language. Next week, we'll meet one such high-level language, namely PYTHON.

1.1.2 Study material and tools

This week's study material can be found on Canvas:

- A document about Boolean logic.
- A document about number representation.
- A document about computer architecture, which is mostly intended as background material; the lecture + slides + your own experience in the pearl assignment should suffice.

Also the slides of both lectures will be put on Canvas.

Besides that, there are reference documents about the Arduino and the processor inside it:

- A document containing its instruction set.
- A very big document about the processor; in principle you don't need it, but it's there just in case.

H	Mon	Tue	Wed	Thu	Fri
1	M lec T	P lec T	M self A	P lec T	M self N
2	M lec T	P lec T	M self A	P lec T	M self N
3	P lec T	P self N	M tut T	P ass A	P ass A
4	F lec T	P self N	M tut T	P ass A	P ass A
6	P lec T	P prac A	P ass A	M tut T	P tst T
7	P lec T	P prac A	P ass A	M tut T	P tst T
8	P tut T	P ass A	P self N	M self N	P ass A
9	P tut T	P ass A	P self N	M self N	P ass A

Abbr	Form
ass	Assignment
lec	Lecture
pfb	Peer feedback
prac	Practical
pj	Project
qa	Q&A
self	Self study
tst	Test
tut	Tutorial

Abbr.	Part
P	Pearl
F	Final project
S	Academic skills
M	Math

Abbr	Supervision
T	Teacher
A	Teaching assistent
N	None

Ma 1–2 Math.
3–4 Lecture. The first lecture gives an overview of the module, both about contents, what is Computer Science and what are you going to learn in this module, and about organisation, what do you have to do, when and who is going to help your with that. The first part of the lecture provides an overview of all of the contents of this module guide. The second part introduces the module's final project.
6–7 Lecture about *binary logic and calculation*. Computers calculate using bits, i.e., zeros and ones. Before we can study (tomorrow) how a computer works, we first have a look at how to calculate with zeros and ones. One aspect of this is the so-called Boolean algebra: a set of rules that are logical if 0 means false and 1 means true. Another aspect is how to represent numbers bigger than 1 and smaller than 0 by a row of bits.
8–9 Tutorial (Dutch: "werkcollege") about *binary logic and calculation*; the exercises are in Section 1.2.1

Di 1–2 Lecture about *computer architecture*. In this lecture we study the parts that just about every computer is made of, and how those parts work together: memory, registers, arithmetic and logic unit, input and output. As an illustration, we use a very simple computer, the so-called Arduino. In this week's pearl assignment, you'll work on this Arduino in practice.
3–4 Self study.
6–7 Start of the Arduino assignment. Each group of two students can borrow one Arduino from us for this week's assignment. Before starting the main assignment, we'll first do some introductory assignments to get acquainted with the Arduino. The assigments are in Sections 1.2.2 and 1.2.3.
8–9 Work on the assignment.

Wo 1–4 Math.
6–7 Work on the assignment.
8–9 Self study, both about binary computation and about computer architecture, if you didn't do this yet. Continue to finish the assignments about binary logic and calculation from Monday.

Do 1–2 Practice test. During the first hour, we do a practice test, and in the second hour we'll discuss the answers. On the one hand this serves as extra practice material, and on the other hand it gives you an idea of how well you understand the subject already. This test is about all topics of this pearl, and it is representative for the real test on Friday. Example questions can be found in Section 1.3.
3–4 Work on the assignment.
6–9 Math.

Vr 1–2 Selfstudy for math.
3–4 Selfstudy for the test / work on the assignment.
6–7 Test.
8–9 Finish the assignment.

Furthermore, we use the following tools to program the Arduino:

- A text editor of your choice (e.g., Notepad).
- A program which loads your Arduino program via a USB cable to the Arduino (`upload.bat`).
- A program called "Putty" in which you can see what the Arduino sends back; this is useful to find errors in your program.
- A program which translates machine languages commands ("mnemonics") into hexadecimal numbers (avra and `asmupload.bat`).

The latter three are in a zip file which you can find on Canvas.

1.1.3 Mandatory sessions and deliverables

Compulsory elements of this week are:

- Participation in the Academic Skills-session.
- Participation in the test.
- Handing in the pearl assignment, and demonstrating it to the teaching assistent.

1.2 Description of the sessions and lab assignments

1.2.1 Tutorial exercises

☞ **1.1** Use Boolean algebra to prove the following formulas:

(a) $1(1+1) = 1$

(b) $\overline{(\overline{A}+\overline{B})} = AB$

(c) $\overline{A}B + A\overline{A} + BB = B$

□

☞ **1.2** Use the result from (b) in the previous exercise to sketch how to build an AND gate from just NOR gates. □

☞ **1.3** Give a realisation of $F = (A \cdot B) + C$ in which only AND gates and inverters are used. Use Boolean algebra to show that your idea is indeed correct. □

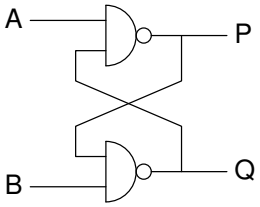
☞ **1.4** Do the following number conversions:

(a) 001 unsigned binary to decimal
(b) 110 unsigned binary to decimal
(c) 110 2-complements binary to decimal
(d) 15 decimal to unsigned binary
(e) -5 decimal to 5-digit 2-complements binary
(f) -5 decimal to 5-digit 1-complements binary
(g) 1D hexadecimal to decimal
(h) 100 decimal to hexadecimal
(i) E9 hexadecimal to binary
(j) 010101100111 binary to hexadecimal

□

☞ **1.5** Consider adding two numbers of one bit each. This results in a two-bit number. Make a truth-table for this, a logical schematic diagram in which you can use any desired type of gate, and a diagram in which you use only NAND gates. □

☞ **1.6** Make a truth table for the following schematic diagram.
Hint: you'll notice that for some combination of input signals, this circuit behaves in a fundamentally different way than the previous circuits; try to describe what this behaviour is.



□

1.2.2 Introductory Arduino assignments

In this pearl, you will be writing small machine language programs for the Arduino. As we've seen in the lecture, such a program consists of a pattern of zeros and ones in the memory of the computer (the Arduino in our case). In the very first computers, one could put those zeros and ones directly into the memory of the computer using switches, but that's not possible with the Arduino. We therefore use a tool. We write the zeros and ones hexadecimally into a text file on a normal PC or laptop, and then upload these to the Arduino using an auxiliary program (available on Canvas). For that hexadecimal text file, of which you'll find an example below at the first assignment, the following rules hold:

- Each Arduino instruction consists of 16 bits, so 4 hexadecimal digits.
- You can put multiple Arduino instructions on a single line, separated by spaces and/or tabs.
- If on a line there is a semicolon (`;`), everything after that is ignored. This is very handy to write text after the hexadecimal code which is readable for humans (called "comments"), like we also do in the example.
- The first hexadecimal number will end up at adress 0 in the Arduino's memory, the second on address 1, and so on. If you want to deviate from this (but that is surely not needed for the introductory assignments), you can write a hexadecimal number preceeded by an exclamation mark (`!`) in the text file: that number will then be used as the starting address for writing the following numbers (without exclamation mark) to the Arduino's memory.

To find out which instructions the Arduino has and what their encoding is, there is a document called "instruction set" on Canvas.

As an example and starting point, we give you the following program:

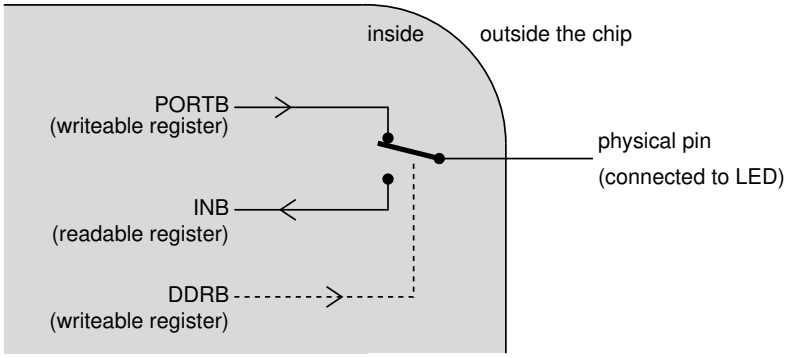
```
e200    ; ldi r16,$20     Register 16 now contains 0010 0000

b904    ; out 4,r16       We write r16 to the Data Direction Register:
        ;                 it makes the pin to which the LED is connected act as an
        ;                 output pin. This needs to be done only once: the pin
        ;                 remains an output as long as the program is running.

b905    ; out 5,r16       Write that same 0010 0000 to the output register; thus
        ;                 the pin is really set to 1 now, making the LED light up.

cfff    ; rjmp -1         Endless loop, don't do anything anymore.
```

Explanation:

- `ldi` means "load immediate", and the dollar sign (in the comments) is a reminder that the number is written in hexadecimal.
- The `out` instruction sends a byte to a special register that controls external hardware; in our case, this external hardware is the LED. These I/O registers are comparable to the normal registers `r0` through `r31`, but have their own numbering (addresses), and have their own instructions for writing to and reading from them.

- The Arduino processor has a number of Input/Output ports[1]. Each I/O port consists of 8 pins (electrical connections) of the processor. Each of those pins corresponds to one bit in a byte that can be written to that I/O port, or read from it. Each pin can be used either as an input (e.g., to read the status of a switch or sensor), or as an output (to drive an LED, buzzer, etc.). Since a pin cannot be used as both an input and an output at the same time, we need to tell the Arduino which pins are to be used as inputs and which as outputs, as will be explained below.

- The first I/O register we use, is officially called (in the processor's documentation) DDRB: "Data Direction Register for port B", and has address 4. The bits in this register determine whether that pin of the processor will be used as an input (in case of 0) or as an output (in case of 1). Typically, the bits in this register would be set somewhere at the beginning of the program, and then remain untouched in the rest of the program.
  In our example program, the binary value `0010 0000` is written to this register, using the `out 4,...` instruction. It will switch one pin to be an output, namely the pin corresponding to the 3rd bit (counting from left). This is the pin the LED is connected to.



- The second I/O register we use, is officially called PORTB, and has address 5. The bits you write to it determine whether the corresponding pin of the processor will have a low voltage (0 volts, if the bit is 0), or a (comparatively) high voltage (5 volts, if the bit is 1). Note that this only works if that pin had previously been made an output by writing a 1 bit to DDRB, as noted above.
  In our example program, we write the binary value `0010 0000` to this register, setting one pin to the high voltage. This pin is the one connected to the LED, thus lighting it up.

- How do we know which of the 8 bits in DDRB and PORTB is the one connected to the LED? For the Arduino, I just told you in the above. In general, one would have to look into the Arduino documentation, or follow the wires under the paint on the Arduino board to find out which pin of the chip is connected to which external hardware.

- `rjmp` means "relative jump": a jump to an address that is calculated with respect to the address where we would otherwise (i.e., without the jump) be. So –1 means jumping back by 1 step, while normally (i.e., without a jump) we'd do 1 step forward (because we normally execute the instructions one by one), so the net result is doing the same instruction again. And again. And again...
  Besides these relative jumps, the processor also has "absolute" jumps, in which the address to which to jump is coded directly, not relative to the current address.

Instructions for use of the Arduino tools on MS-Windows:

- Unpack the ZIP file from Canvas; this creates a new directory (called "`module 1.1 week 1`") with some contents. That new directory will henceforth be called your working directory. It is also in this directory that you should put the programs (hex files) you write yourself.

[1] In Dutch, the word "poort" is used both for this kind of I/O ports, and for logic gates (AND, OR, NAND etc.). In English, the words are different.

- Install `arduino-1.6.11-windows.exe`
- Install `WinAVR-20100110-install.exe`, but **watch out** while doing this, since by default this installer messes up the PATH variables. Therefore, proceed as follows:
  - During the installation of WinAVR:
    - Remember the directory where WinAVR was installed, the default is: `C:\WinAVR-20100110`
    - On the screen "Choose Components" **de**select "Add Directories to PATH (Recommended)". (This option will break it.).
  - After the installation:
    - Start SystemPropertiesAdvanced.exe (For example using [Win]+R).
    - Click "Environment Variables..", the button in the bottom right corner.
    - In the list below "System variables" find the variable "PATH" and click "Edit..."
    - Now add `C:\WinAVR-20100110\bin;C:\WinAVR-20100110\utils\bin;` if you installed WinAVR in `C:\WinAVR-20100110`; otherwise you will have to replace `C:\WinAVR-20100110` with the installation directory. ! Beware to not break the path yourself !
- Connect the Arduino.
  - Windows 7/8/10: Open Start > Apparaten en printers (or Devices and Printers). Look for "Arduino (COMnr)" and remember the COM port number.
  - Windows Vista and older: Open "Apparaatbeheer" (right mouse button > Deze Computer > Eigenschappen > Apparaatbeheer). (or This Computer > System Properties > Device manager) Look for "Arduino (COMnr)" and remember the COM port number.
- Open a command prompt (cmd.exe) in the working directory. In case you're unfamiliar with this, there are several ways to do this:
  - In Windows 8/10:
    Go to your working directory in Explorer, and choose "Open command prompt".
  - Or, in Windows 7:
    Go to your working directory in Explorer, keep Shift pressed while right-clicking on the folder, and choose "Open command window here".
  - Or, just start `cmd.exe`; then, at the command prompt, use the `cd` command to navigate to your working directory.
- To upload hex files to the Arduino:
  Given a hex file `demo.txt` and the Arduino connected to COM13, type at the prompt:
  `upload com13 demo.txt`.

Instructions for use of the Arduino tools on MacOS:

- Please follow the instructions on `https://github.com/RemcodM/pearl000-files`

Instructions for use of the Arduino tools on Linux:

- Please follow the instructions on `https://github.com/RemcodM/pearl000-files`.

☞ **1.7** Type the above program in Notepad (or another editor), and save it as `prog7.txt`.[2] Upload the program

[2] It would be more logical to let the filename and in `.hex` or something like that, but Notepad has the nasty habbit of adding another `.txt` if you try that.

to the Arduino, and check that it works.
Look up the three instructions from the above example program in the instruction set document (available on Canvas), compute their hexadecimal codes, and compare those to the codes in the program (this may seem useless now, but you'll need it later).
Hint: the instruction set document is big, but highly structured. You'll need it a lot in this pearl, so it's good to familiarize yourself with it. The first 15 pages can be skipped. After that, pages 16 through 157 describe all instructions of the Arduino processor in detail, in alphabetical order of their mnemonics. (In Adobe Acrobat, you can enable an index via View → Show/Hide → Naviation Panes → Bookmarks.) Go for example to page 94, about the `LDI` instruction (the first in our example program). It tells you schematically what this instruction does (assign K to register d). It also gives you the "Opcode", the binary code representing this instruction, which has some 0s and 1s, and letters `K` and `d` which need to be substituted by the constant K and the register number d. The page gives even more information, which you'll need later on. □

☞ **1.8** Change the program such that it switches the LED off; save this to a file called `prog8.txt`, and check that it works.
Note: for all programs, please always write a comment after every instruction with its mnemonic! This is not only handy for yourself, but also for the teachers and teaching assistants. □

☞ **1.9** Make a program called `prog9.txt` which does the following: switch the LED on, then switch the LED off, and then jump back go switching it on; basically, let it flash the LED very fast.
At the beginning of this program, put the following codes:

```
e0f0 e6e1 e800 8300 e008 8300
```

These instructions tell the processor to run slower: normally it does 16 million instructions per second, but after this line it runs 256 times slower (until it is reset). We need this slowdown because on some Arduinos the LED otherwise cannot keep up with being switched on and off so fast. (In practice, a good reason for slowing down a processor could be to reduce its power consumption.)

Calculate how quickly the LED will be switched on and off, knowing that the Arduino now runs at $\frac{16}{256}$ million clock cycles per second; what do you expect to see from this with the naked eye?

Let a TA sign off assigments 1.8 and 1.9.

□

When programming the Arduino, it would be handy to peek into the chip to see the contents of the registers, especially if it doesn't do what you expect it to do. Some of the earliest computers indeed had lamps (LEDs hadn't been invented yet) on their front panel showing the state of all the bits in their registers, but the Arduino is too small for that. Therefore, we (the teachers) have made a small program which you can use to send the contents of `r16` back to your laptop via the USB cable. To use this, put the following code in your Arduino program:

```
940e 0200    ; send r16 to laptop
```

Instructions for viewing the data sent back to the laptop by the Arduino:
- Upload the program containing the above hex codes to the Arduino.
- Open Putty.exe. Choose the "Serial" connection type and the right COM port.
- Click "Open". The Arduino will restart and as soon as the above line of your program is reached, the content from `r16` will be sent to the laptop and displayed by Putty (in binary).
- Close Putty again before you upload a new program to the Arduino, otherwise the upload is disturbed (since Putty and the upload tool try to receive from the Arduino simultaneously).
- For Linux users: use cutecom instead of putty.

☞ **1.10** Try this: write a program called `prog10.txt` which writes the number 3B hex into r16 and sends it to your laptop. Verify that the bits your laptop shows indeed match 3B hex. □

☞ **1.11** Experimentally verify that the `subi` instruction indeed uses 2-complement notation for negative numbers. □

Let a TA sign off assigments 1.10 and 1.11.

By now, you're probably fed up with translating the instructions into hexadecimal codes by hand, and having to count precisely where the `rjmp` goes. It's good to have seen once how this works, but it is not a handy way to write large programs.

That's why, already quite early in the history of computing, so-called *assemblers* have been created: computer programs which can read the mnemonics and convert them into the corresponding binary or hexadecimal codes. From now on, you're allowed to use the assembler which you can find on Canvas. As an example, we give the example program from page 20, but now in assembly (that's what the assembler's language is called) format:

```
.device atmega168      ; what type of processor do we use? don't change!

.equ DDRB = 4          ; address of data-direction register; don't change!
.equ PORTB = 5         ; address of output port register; don't change!

   ldi r16,$20         ; register 16 now contains 0010 0000
   out DDRB,r16        ; write this to the data-direction-register to make
                       ; the pin with the LED act as output
   out PORTB,r16       ; and also write to the port itself, which switches
                       ; the LED on

again:                 ; this is not an instruction, but a label, as
                       ; indicated by the colon at the end

   rjmp again          ; infinite loop: jump back to the label
                       ; note: do not use rjmp -1 anymore, only use labels!
```

Note how the assembler makes our life easier:

- we no longer need to lookup the hexadecimal codes for the instructions;
- we can give names to numbers, as was done in the example for `DDRB` and `PORTB`, which makes mistakes less likely and makes the program more readable (since these represent properties of the processor, you should **not** change them from the values given above);
- jumps are computed automatically: we can give names to places in the program, like `again` in the example, and use that name in the jump instruction.
  Note: always use labels; do not try to use numbers behind `rjmp`, since the assembler will try to interpret such numbers as absolute addresses and incorrectly convert them to relative offsets for you.

And this is just the start. When using an assembler like this, there is still a one-to-one correspondence between what you type and the individual instructions given to the processor. The next step up is the use of a "higher-level programming language", in which that correspondence is no longer there; one example of such a language is Python, which we will use next week. Computer scientists and engineers have created increasingly more advanced tools to allow describing what a program should do at increasingly high levels of abstraction (and thus easier and with fewer errors). However, it is good to keep in mind that, with whatever tools you create programs, in the end they will be converted to machine language instructions, because those are the only ones that the computer can really process.

Instructies for using the assembler tool:

- To upload assembly programs to the Arduino:
  Given the assembly file `demo.txt` and the Arduino connected to COM13, type at the command prompt: `asmupload com13 demo.txt`.
- Reading a register from the Arduino now works differently:
  - Instead of the hex codes given previously, put the following line into your assembly program at the point where you want to send `r16` to the laptop: `call sendr16tolaptop`
  - At the **eind** of your asm file, insert the following line: `.include "rs232link.inc"`

Brief instructions for (Ubuntu) Linux:

- Install avra: `apt-get install avra`
- Make your own equivalent of the `asmupload.bat` contained in the zip file for Windows.

☞ **1.12** Put the above program into a file called `prog12.txt`, upload it via the assembler to the Arduino, and observe that it works. Then modify the program to switch off the LED instead of on (like assignment 1.8).
Let a TA sign off this assigment. □

1.2.3 Main Arduino assignments

☞ **1.13** Add so much delay to your previous LED program, that the LED flashes at a rate which is visible for humans. Note that you can't do this simply by adding nops, because the Arduino doesn't have enough memory for so many nops (millions!). So you'll have to think of something else.
Hint: try to build and test parts of your solution, instead of hoping to get it all correct at once.
Let a TA sign off your work. □

Besides the LED, you can also connect a buzzer to the Arduino. Do this by inserting its black wire into the connection marked "GND" and the red wire into the connection marked "12". The buzzer won't buzz by itself though. Your program should make pin 12 an output (like you already made the LED pin an output); this pin corresponds to bit 0001 0000 in the DDRB- and PORTB-registers. Next, you have to switch this pin between 0 and 1 at a frequency which corresponds to an audible tone (e.g., about 1 kHz).

☞ **1.14** Set yourself an aim w.r.t. the use of the LED and/or the buzzer, e.g. letting the LED flash in a morse-code pattern, make the buzzer make a siren-sound or even play a Beethoven symphony. Discuss your plan with the teaching assistant, and realise it.
Hint: again, don't try to write a working program for your final goal all at once, but build in steps, where you test each step separately (with LED and buzzer, and/or by sending data back to your laptop).
Let a TA sign off your work. □

Bonus points

In this pearl, two half bonus points can be earned.

Half bonus point: music

One half bonus point can be earned from assignment 1.14 by having the Arduino play a *clearly recognizable melody* of at least 12 tones, and making good use of the `call` instruction (look up for yourself what this does). Whether the melody is recognizable, is to be judged by the teaching assistant.

Half bonus point: multiplication

The other half bonus point can be earned by writing a program which multiplies two 4-bit (unsigned) numbers, resulting in an 8-bit unsigned number, *without* using the processor's multiply instructions (`mul`). Also, it is not allowed to simply do a repeated addition, because that would be inefficient for large numbers. Think of you how you learned to do multiplications of decimal numbers in primary school, and translate

that to binary numbers. You may want to use the processor's *rotate* and/or *shift* instructions. The program should expect the two numbers to be multiplied in registers `r17` and `r18`, and deliver the result in `r16`, so it can be sent to your laptop for testing.

Note that in this bonus assignment, the two aspects of this pearl (binary calculation and machinecode programming) meet.

Your bonus depends on how short your program is: the shortest program (fewest instructions) submitted will get the entire half point, longer programs get a proportionally smaller bonus. Hint: it's possible to do this in less than 10 instructions! We count only those instructions needed to do the calculation. (Your program should *start* with two `ldi`'s to fill `r17` and `r18` with some test numbers to be multiplied; these two instructions are *not* counted. At the end, after your code for doing the multiplication, you put a `rcall` to send the result to the laptop; this `rcall` is also *not* counted, nor any instructions *after* this (e.g., an infinitely loop to end the program). All other instructions *are* counted.)

Let a TA sign off your work.

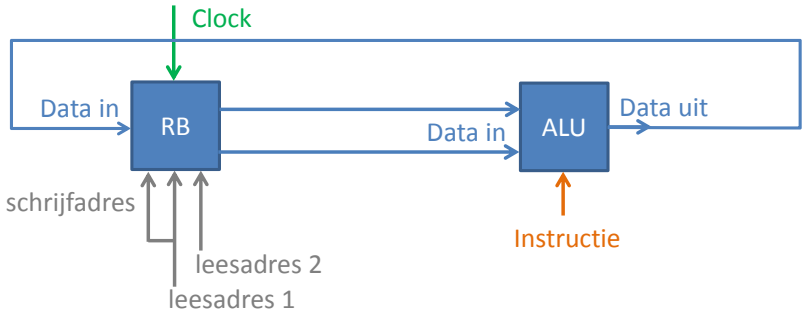
Handing in

You can hand in your work via Canvas. Hand in your assembly files for assignment 1.14, and if applicable, also the assembly file for the second bonus question. Each file should have sufficient explanations ("comments") to make clear to the teachers how your program works. If you hand in multiple files, please do *not* put them together in a `.rar` or `.zip` archive.

1.3 Example test questions

The test will be both about binary arithmetic and logic, and about computer architecture.
Example questions about binary arithmetic and logic can be found in Section 1.2.1. Example questions about computer architecture are given below.

1.



[Translation: uit = out, schrijfadres = write address, leesadres = read address, instructie = instruction]

The above simple processor has two instructions: 0 = '+' (add) and 1 = '*' (multiply).

Give the program for this processor to do the following computation: R2 = (R1+R2)*(R0+R1)

	read adress 1 / write address	read address 2	instruction
Time slot 0			
Time slot 1			
Time slot 2			
Time slot 3			
Time slot 4			
...			

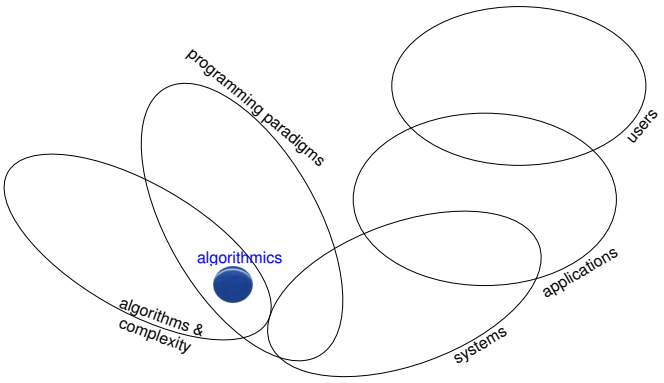
2. Given the following AVR program ("BRNE" means "BRanch if Not Equal", "MUL" means MULtiply, "DEC" means "DECrement (subtract 1)" and "SUB" means "Subtrac t"):

```
LDI  R16, $03
LDI  R17, $03
LDI  R18, $02
LDI  R20, $01
MUL  R17, R18
DEC  R16
MOV  R19, R16
SUB  R19, R20
BRNE -5
```

   Give a table showing how the contents of the registers changes while this program is executed; you can choose to use either hexadecimal or decimal numbers, but do show which of the two you choose.

   How many clock cycles are needed for the execution of this program?

The Blue Pearl is the subtle abode of the inner Self
Swami Muktananda —Does Death Really Exist?



Week **2**

Pearl 001: Algorithmics

2.1 Introduction

To make a computer do what you want it to do, you have to program it. A lot of issues are involved, but one of the main requirements is to have a good insight in the *algorithms* you can use in your programs. In this pearl you will meet a number of existing, fundamental algorithms for *sorting* and *searching*.

The working of an algorithm is independent of the programming language you use, but to practice we have to choose a language. For this week, the choice has fallen on PYTHON (version 3).

Last week you have also programmed a bit, but on a much lower level, where you had to include every single step the processor should take. PYTHON is a "higher-order language", which makes the life of a programmer a whole lot easier.

2.1.1 Global description of the pearl assignment

The pearl assignment will consist of programming a search engine: after typing in a search term, you will get a list of (names of) documents in which this term occurs. The assignment can be extended at will, for instance by returning the results in decreasing order of frequency, by allowing more complex search terms, or in some other way you may think of yourself. You will do the assignment in pairs.

2.1.2 Study material and tools

The study material of this pearl consists of:

- Slides and this reader: on Canvas
- Tutorial PYTHON intro slides: on Canvas
- Tutorials and language definition of PYTHON, e.g., `https://docs.python.org/3.6/tutorial`

The practical of this pearl uses the following tools and files:

H	Mon	Tue	Wed	Thu	Fri
1	M tst T	P lec T	M self A	P ass N	M self N
2	M tst T	P prac N	M self A	P ass N	M self N
3	M lec T	S tut T	M tut T	P ass A	P ass A
4	M lec T	S tut T	M tut T	P ass A	P ass A
6	P lec T	P prac A	P ass A	M tut T	P tst T
7	P prac A	P prac A	P ass A	M tut T	P tst T
8	P prac A	P prac A	S lec T	M tut T	P ass N
9	P prac A	P prac A	S lec T	M tut T	P ass N

Abbr	Form
ass	Assignment
lec	Lecture
pfb	Peer feedback
prac	Practical
pj	Project
qa	Q&A
self	Self study
tst	Test
tut	Tutorial

Abbr.	Part
P	Pearl
F	Final project
S	Academic skills
M	Math

Abbr	Supervision
T	Teacher
A	Teaching assistent
N	None

Mon
- 1–4 Math
- 6 Introductory lecture: What are algorithms? What can you do with them? Why is it important to know about them? Why PYTHON?
- 7–9 Practical: linear and binary search in PYTHON (Sections 2.2.1–2.2.2)

Tue
- 1 Introductory lecture: Sorting, fast and slow. What makes (searching and) sorting important and interesting?
- 2 Practical (self-study): Various sorting algorithms in PYTHON (Sections 2.2.3–2.2.4)
- 3–4 Academic skills
- 6–9 Continuation of practical 2nd hour (supervised)

Wed
- 1–4 Math
- 6–7 Starting the pearl assignment (Section 2.2.5)
- 8–9 Academic skills

Thu
- 1–2 Working on pearl assignment (self-study)
- 3–4 Working on pearl assignment (supervised)
- 6–9 Math

Fri
- 1–2 Math self-study or pearl test preparation
- 3–4 Finishing pearl assignment (supervised)
- 6–7 Test
- 8–9 Finishing pearl assignment (self-study)

Week 2 per session

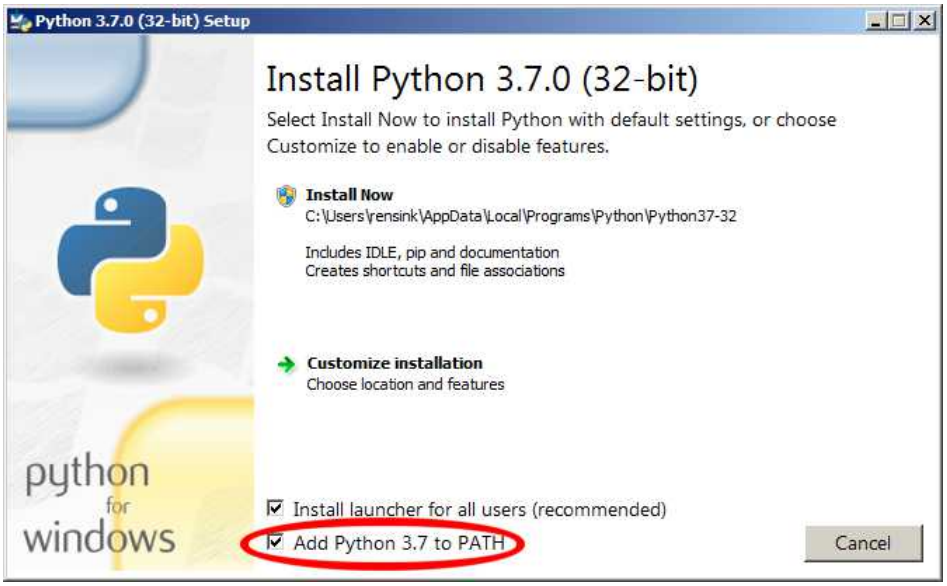


Figure 2.1: Tick the "Add Python 3.7 to PATH" box during installation

- A PYTHON 3 interpreter, preferably the newest version (3.7.0 at the time of writing). *Nota bene:* when installing the interpreter under Windows, the `PATH` variable should be set. For this purpose, during installation, *tick the box "Add Python 3.7 to PATH"* (see Figure 2.1).
- An arbitrary text editor; for instance `IDLE` that is packaged with PYTHON, but alternatively `NotePad++` is an excellent choice as well.
- Pre-defined PYTHON-files (on Canvas)
- Pre-defined text files for the practical exercises and pearl assignment (on Canvas)

You are entirely free to use the wealth of material on the Internet if and when you need more (practice) material on PYTHON. There are many, many beginner courses, with many, many examples and other material. Of course, your solutions for the practical exercises and pearl assignment should be your own; never just copy and paste.

2.1.3 Mandatory sessions and deliverables

Mandatory elements of this week are:

- Having the practical exercises signed off
- Participating in the Academic Skills-sessions
- Participating in the test
- Handing in the pearl assignment

2.1.4 What constitutes fraud?

When it comes down to handing in the assignment of this pearl, every year there are students who do not understand the borderline between, on the one hand, cooperating and discussing solutions between groups (which is allowed), and on the other, copying or sharing solutions (which is forbidden and counted as fraudulent behaviour). Here are some scenarios which may help in making this distinction.

- **Scenario.** Peter and Lisa are quite comfortable with programming and have pretty much finished the assignment. Mark and Wouter, on the other hand, are struggling and ask Lisa how she has solved it. Lisa, a friendly girl, shows her solution and takes them through it line by line. Mark and Wouter think

they now understand and go off to create their own solution, based on what they saw. *Is this allowed or not?*

Verdict. No problem here, everything is in the green. It is perfectly fine and allowed for Lisa to explain her solution, even very thoroughly. The important point is that in implementing it themselves and testing their own solution, Mark and Wouter are still forced to think about what is happening and will gain the required understanding, though probably they will not get as much out of it as Lisa (explaining stuff to others is about the best possible way to learn it better yourself!)

- **Scenario.** The start is as in the previous case. However, while Mark and Wouter implement their own solution, inspired by that of Lisa, some error crops up which they do not understand. Lisa has left by now; after they mail her, still trying to be helpful she sends them her solution for them to inspect. They inspect it so closely that in the end their solution is indistinguishable from Peter and Lisa's, except for the choice of some variable names and the comments they added themselves. *Is this allowed or not?*

  Verdict. This is now a case of fraud. All three are at fault: Lisa for enabling fraud by sending her files (even if it was meant as a friendly gesture) and Mark and Wouter for copying the code. Peter was not involved, developed his own solution (together with Lisa) and is innocent.

- **Scenario.** Alexandra and Nahuel are not finished, and the deadline is very close. The same holds for Simon and Jaco. On the Friday night train home, Jaco and Nahuel meet and during the 2-hour train ride work it out together. They type in the same solution and hand it in on behalf of their groups. *Is this allowed or not?*

  Verdict. This is also a case of fraud. Actually there are two problems here. The first is that both Nahuel and Jaco handed in code on behalf of their groups that had been developed by them alone, without their partners. This is unwise and against the spirit of the assignment (Alexandra and Simon also need to master this stuff!) but essentially undetectable and not fraudulent. The second problem is that the solution was developed, and shared, in collaboration between two groups; this is definitely forbidden. All four students are culpable; Alexandra and Simon cannot hide behind the fact that they did not partake in the collaboration, as they were apparently happy enough to have their name on the solutions and pretend they worked on it, too.

Note that we are not on a witch-hunt here: let us stress again that cooperating and discussing assignments is OK, even encouraged; it is at the point where you start copying or duplicating pieces of code that you cross the border.

2.2 Description of the sessions and lab assignments

The right way to do the exercises of this practical is:

Read the entire text and not just the exercises. The text in between the exercises serves a purpose: it contains explanations and hints. Make sure you understand everything you read, ask for help otherwise.

Do all exercises that are not marked "optional". Not everything will be checked or signed off; it is in your own interest to also do those parts that are *not* signed off. These are often (kinds of) questions that may also be asked in the test.

Have everything at hand when having your solutions signed off, including tests you ran and the resulting outcomes.

2.2.1 Linear search

Suppose you get a printed list of student numbers, and you want to check if your number is on it. How would you do this? The most obvious way is to scan the list from beginning to end and to compare every element of the list with your own number, until you have found the number or you have reached the end of

the list. This is called *linear learch* (because you through the list in a straight line from beginning to end), and it is an example of an *algorithm*.

Writing a computer program largely consists of choosing and implementing algorithms, but the algorithms are themselves independent of the programming language you happen to be using. One can in fact give a "pure" description of an algorithm, without sticking to any actual programming language: this can aid understanding because the incidental syntax and design choices of the language do not play a role any more. This leads to so-called *pseudo-code*. Algorithm 1 shows the pseudo-code for linear search.

Algorithm 1: Linear search in an arbitrary list

input : List *data* and value *val*
output: The index of *val* in *data*, or -1 if *val* does not occur in *data*

```
1 Assign the first valid index of data to i;
2 while i valid index in data and data[i] ≠ val do
3     Increment i by one;
4 end
5 return If i is not a valid index in data then −1, otherwise i
```

To understand Algorithm 1, you have to know what is meant by a *list* and an *index*. A list is essentially than a sequence of values — for instance, numbers or words. Those values are then numbered in the order they occur in the list; this number is called the *index*. For historic reasons, in PYTHON numbering starts at 0 (rather than at 1).

Below you see an example list consisting of 8 words, with corresponding indexes 0 through 7. If we call the entire list *data*, then $data[0]$ stands for the first element ("For"), $data[1]$ for the second ("historic"), etc. In general, $data[i]$ is the i-th element, where $0 \leq i < 8$. An index is called *valid* if it is in the range from 0 through 7. Note that, when writing $data[i]$, the index is surrounded by *square brackets*, and not the more usual parentheses — we will meet those later as well.

Value	***Index***
For	*0*
historic	*1*
reasons	*2*
numbering	*3*
often	*4*
starts	*5*
at	*6*
0	*7*

☞ **2.1** Study Algorithm 1 until you completely understand it, for instance through a set-by-step calculation of what happens if you search for the words "often" and "never" in the list above. What result does the algorithm return for those cases? □

In the next exercises, your task is to program the linear search algorithm. As announced, in this pearl we use the programming language PYTHON, but we will use only a tiny part of this language. If you want to know more of the language, you can find all you want to know on the Internet — a good place to start is the Wikipedia page of PYTHON.

Please note: if you already know PYTHON well, you probably know there are a lot of shortcuts that make all the things we do here much easier. *Do not use those shortcuts* and stick to the fragment of PYTHON presented in this pearl. The advanced language features partially or completely hide the algorithms that this pearl is about, so using them defeats the learning goals.

The practical exercises make use of a number of pre-defined PYTHON files (so-called "modules"). These are text files with the file extension `.py`, which are treated as separate units by the PYTHON interpreter.

The following text fragment from the pre-defined file `search.py` (to be found on Canvas) is a PYTHON

function for linear search; in other words, it is the PYTHON version of the pseudo-code in Algorithm 1. A *function* is the appropriate way to package such an algorithm and give it a name. After defining a function, one can use the name to *call* the function, so that the algorithm is actually executed.

```
1  def linear(data, value):
2      print("Modified!")
3      """Return the index of 'value' in 'data', or -1 if it does not occur"""
4      # Go through the data list from index 0 upwards
5      i = 0
6      # continue until value found or index outside valid range
7      while i < len(data) and data[i] != value:
8          # increase the index to go to the next data value
9          i = i + 1
10     # test if we have found the value
11     if i == len(data):
12         # no, we went outside valid range; return -1
13         return -1
14     else:
15         # yes, we found the value; return the index
16         return i
```

To understand this piece of code, you need to know the following about PYTHON:

- **`def`** (line 1) starts a function definition, of a function named `linear`. The names between parentheses, `data` and `value`, are *parameters* of this function: they correspond to the data structures this function can use to do its job.
- The second line (starting and ending with three quotes) describes the purpose of the function. This is documentation meant for a programmer who wants to *use* (i.e., call) the function; the computer completely ignores this line during execution.
- Lines starting with `#` are comments. These are meant for a programmer who wants to *understand the code* of the function; these lines are also ignored during execution.
- A line of the form `var = expression` (lines 4 and 8) is an *assignment* and means "`var` *becomes* `something`": `var` is the name of a *variabele* which receives the value of the `something` (see below for a short explanation of what a variable is).
- A line of the form **`while`** `condition`: followed by a block of more deeply indented lines means that, if `condition` holds (i.e., is true), the whole block is executed; this is repeated again and again until `condition` does *not* hold.
- A line of the form **`if`** `condition`: followed by a block of more deeply indented lines means that, if `condition` holds, the whole block is executed, but *in contrast to* ***`while`*** just a single time.
- A line of the form **`else`**: followed by a block of more deeply indented lines belongs to the **`if`** just in front (on the same level of indentation as the **`else`**); the block is executed if the `condition` of that **`if`** did *not* hold.
- A line of the form **`return`** `expression` means that the function is finished, with the value of `expression` as its outcome.
- The operators "`<`" and "`!=`" (line 6) and "`==`" (line 10) each compare two values: the first variant is true if the value to the left of the operator is smaller than that to the right; the second variant is true if the values are not equal, the third if they are equal. *Do not confuse the operator "==" with the single equality symbol, "=": the latter is used for assignment, see above!*

In general, a *variable* is a name chosen by the programmer, which has an associated value. In PYTHON, a new value is associated to a variable by an assignment to that variable, of the form `var = expression` (read: "`var` *becomes* `expression`"), as discussed above. An example is `i = 0` in line 4: after executing this line, the variable `i` has the associated value `0`. Moreover, a variable can be *used*, in an *expression* such as `data[i]` in line 6; here, `data` and `i` are both variables, the first of which is associated with a list and the second with a number: the expression as a whole stands for the value at index `i` of the list `data`, again as discussed above.

In the function above there are altogether three variables: `data`, `value` and `i`.

`data`: a list of elements (numbers, words, ...) in which we are searching. As explained above, the elements of a list have an index, starting at 0 for the first element. The element with index `x` can

be accessed by `data[x]` — again the index should be surrounded by square brackets. The length of the list (i.e., the number of elements) can be queried by **`len`**`(data)`. Note that this uses parentheses rather than square brackets; if we would write **`len`**`[data]`, this would mean that we expect **`len`** to be a list and `data` an index expression![1] It follows that the last element of a list always has index **`len`**`(data)-1`.

`value`: the value to be looked up in the list. If `data` is a list of words, `value` also has to be a word; if `data` is a list of numbers, `value` has to be a number; et cetera.

`i`: the index in `data` where the search currently takes place. This index is initialised to `0` (line 4), and subsequently incremented (line 8); if it is larger than or equal to **`len`**`(data)` it is no longer a valid index (lines 6 and 10).

☞ **2.2** Execute the following on your computer, in one of the following ways:

- Start up IDLE (which is part of the PYTHON installation), open the `search.py` module (File → Open...) and in the editor window run the module (Run → Run Module)
- In a Windows explorer, select the folder where `search.py` is located, then File → Open Windows Powershell → Open Windows Powershell.
- Open a command-line window (On Windows: Start → type "PowerShell") and change the working directory to the place where `search.py` is located (using `cd`); then type `python` (under Windows) or `python3` (under Mac OS or Linux).

In all of these cases, you should now be facing a window with more or less the following text:

```
Python 3.7.0 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) ...
Type "copyright", "credits" or "license" for more information.
>>>
```

Especially watch the version number (here `3.7.0`): if this is `2.x.y` then you have the wrong version of PYTHON. The part `>>>` is the so-called PYTHON *prompt*: here is where you type your commands.

(a) Try out the function `linear`. For instance, type the following successive lines at the PYTHON prompt:

```
import search
search.linear([4, 2, 3, 2], 2)
search.linear([4, 2, 3, 2], 5)
search.linear(["a", "short", "sentence"], "sentence")
search.linear(["a", "short", "sentence"], 2)
```

The first line imports the module `search` containing the function `linear`, which has the effect that from that moment onwards, that function is known to the interpreter and can be used; the next lines each time call `linear` with different lists and search values.

(b) Explain the output of the interpreter in the previous step.

□

Terminology

Definition, call. Functions such as `linear` are first *defined* and then *called*.

Parameters. In both a definition and a call, the function name is followed by parentheses surrounding a — possibly empty — comma-separated list of names (in case of a definition) and expressions (in case of a call); these are called *parameters*.

Formal parameters. In a definition, the parameters (in that case sometimes called *formal parameters*) are always *variables*, such as `data` and `value` in the case of `linear`).

Actual parameters, arguments. In a call, the parameters (in that case sometimes called *actual parameters* or *arguments*) can be arbitrary expressions, such as `[4, 2, 3, 2]` and `2` in `2` but also `data` in **`len`**`(data)`.

It is one thing to understand a function that has already been defined, but another to modify it or write a function yourself. We will start with a few modifications in `linear`.

[1] In fact, **`len`**`(data)` is a call to a function named **`len`**, with `data` as argument. In other words, **`len`** is the same kind of thing as the function `linear` we just defined, but this particular function is already defined in PYTHON by default.

☞ **2.3** Using a text editor, change the function `linear` in the ways described below. To use the modified module (i.e., PYTHON file) in your running interpreter, you first have to type

```
import importlib
importlib.reload(search)
```

(`importlib` is a standard module of PYTHON that defines the function **`reload`**.) Apply the following changes, and test their effect:

(a) Change the outcome in case the searched value does not occur in the list into `'does not occur'`.
(b) Start looking at index `1` instead of `0`. Obviously, this introduces an error; what exactly goes wrong?
(c) Continue up to *and including* **`len`**`(data)` (using `<=`) rather than *up to* (using `<`). What error does this introduce?
(d) Search the list in reverse order. Does this always yield the same result?

□

If you are getting bored having to type `search.linear` each time, instead you can use

```
from search import linear
linear([4, 2, 3, 2], 2)
```

However, you then have to remember to repeat ***`from`*** *`search`* ***`import`*** *`linear` every time you reload `search`! If you don't, then `linear` will still be bound to the previous, unchanged version of the function, though `search.linear` will use the new one.*[2]

When you have executed the same sequence of commands a few time, this too starts to be boring. You can alternatively write a PYTHON script to test the function `linear`, and invoke this script rather than the interpreter.

☞ **2.4** In a text editor, create a file `searchtest.py`, containing precisely the commands you used above to test `linear`, but now as arguments to the standard PYTHON function **`print`**: for instance

```
from search import linear
print(linear([4, 2, 3, 2], 2))
print(linear([4, 2, 3, 2], 5))
print(linear(["a", "short", "sentence"], "sentence"))
print(linear(["a", "short", "sentence"], 2))
```

(The **`print`**-function ensures that the result of calling `linear` is actually sent to the command line window.) Invoke this script from the command line using "`python searchtest.py`".

If everything went right, you can now see the output of the commands in the script, but not the commands themselves. This is not always optimal for understandability. You can also use **`print`** to add some explanatory text to the output; for instance

```
from search import linear
print("Search 2 in [4, 2, 3, 2]; expected outcome = 1")
print(linear([4, 2, 3, 2], 2))
```

When invoking the script, you can now immediately see if the result of the function meets the expectated value. □

2.2.2 Binary search

If you would look for a word in an arbitrary book by a linear search, this would last forever. No one does that. But in case of a dictionary or telephone guide (if you know what those are) it used to be the whole purpose to find a word or name quickly! Fortunately, a lot of profit can be had because the words in a

[2] See `https://stackoverflow.com/questions/1739924/python-reload-component-y-imported-with-from-x-import-y` for a longer explanation of this.

dictionary and the names in a telephone guide are *alphabetically ordered*. In an ordered list you can do a *binary search* instead of a linear one, which is much more efficient, as we will see.

Before going into this: even linear search can be improved if you assume that the list `data` is ordered. To wit, if at any point you notice that the element at `data[i]` is *larger* than the looked-for `value`, you can conclude that `value` does not occur in the list and you can stop searching!

☞ **2.5** Create a new module `ordsearch`, again containing a function `linear`, which is improved in the way described above. Test this function by writing a script `ordsearchtest`. Also show (in that test script) that the function can give wrong results if you use it on an argument list that is not ordered. □

☞ **2.6** What happens if you use `ordsearch.linear` to search for a number in a list of words, or vice versa? Explain the observed effect. □

Although linear search in an ordered list is a light improvement with respect to an unordered list (in case the searched word is not in the list), this does not really help much when the list contains a hundred thousand elements, as a dictionary might. Other solutions are fundamentally more efficient.

☞ **2.7** Imagine how you yourself would look for a word in a dictionary (or a name in a telephone book), and formulate this as an algorithm, in natural language. Use numbered lines to describe the steps in your algorithm, and when needed use terms such as "go to line x". □

When searching in an ordered list, you can think of the *interval* within the list (given by a lower and upper index bound) within which the target value is certain to lie, if it is in the list at all. Initially, the lower bound and upper bounds are simply the lowest and highest indices in the list (which are those?). The interval is each time made shorter by *choosing the middle* and inspecting the value at that index. (If the interval contains an even number of elements, "the middle" can be either of the two central ones; it does not matter which one.) This inspection can result in any of three possible outcomes:

1. The element in the middle is exactly the target value; then we are done.
2. The element in the middle is larger than the target value; then the upper bound should be adjusted to the index just before the middle.
3. The element in the middle is smaller than the target value; then the lower bound should be adjusted to the index just after the middle.

After the upper has been adjusted (step 2), it may be the case that it has become smaller than the lower bound; and vic versa for the adjustment of the lower bound (step 3). In that case the interval within which we are searching has actually become *empty*, meaning that the tareget value does not occur in the list at all.

The above described the so-called *binary* search algorithm, called thus because there are always two ways to continue searching — left of the middle or right of the middle. In Algorithm 2 you will find the pseudo code of this algorithm.

Algorithm 2: Binary search in an ordered list

input : Ordered, non-empty *data* and target value *val*
output: The index of *val* in *data*, or -1 if *val* does not occur in *data*

```
1  Assign to low and high the first and last index in data;
2  while interval between low and high non-empty and data[low] ≠ val do
3      Choose mid in the middle between low and high ;
4      if data[mid] = val then
5          Assign to low the value of mid ;                          /* Found!  */
6      else if val < data[mid] then
7          Assign to high the value of mid − 1;          /* val lies before index mid  */
8      else
9          Assign to low the value of mid + 1;           /* val lies after index mid  */
10     end
11 end
12 return if data[low] = val then low, otherwise −1
```

☞ **2.8** Add a function `binary` to the module `ordsearch`, containing your encoding of the binary search algorithm according to Algorithm 2 in PYTHON. Test your implementation.

Nota bene. To program line 3 of Algorithm 2 you should take the average of *low* and *high*; this involves a division by 2. If you divide an odd number by 2 (for instance by typing `n/2` where `n` equals 5) this will result (in PYTHON) in a real number (in this example 2.5). You cannot use that as an index value; instead, you need to truncate it (in this example to 2). To get the truncated value you can either use `n//2` (where `//` is the PYTHON operator for integer division) or **`int`**`(n/2)` (where **`int`** is a standard PYTHON function that truncates a real number to an integer number). □

☞ **2.9** Suppose you have a list of 200 elements, and you look for a value that eventually turns out to occur at index 62. Which are the values assigned to *low*, *high* and *mid* before the element is found? □

To test your algorithm more extensively, the pre-defined files of this practical include a dictionary `Unabr.dict` containing more than 200000 words. In order to use this file as the first parameter of `linear` or `binary`, it should first be converted to a list usable by PYTHON. For this purpose, the pre-defined files also include a module `util.py` containing two functions:

- `lines(filename)`: returns a list containing all individual lines in the file
- `words(filename)`: returns a list containing all individual words in the file

You can now for instance type (in the interpreter):

```
>>> from ordsearch import binary
>>> from util import lines
>>> binary(lines("Unabr.dict"),"eagle")
-1
>>> binary(lines("Unabr.dict"),"zygose")
213492
```

(Check that "eagle" indeed does not occur in the dictionary!)

We have already claimed that binary search is more efficient than linear search. That claim should be supported both analytically and experimentally. For this purpose we take the most time-consuming case, namely that the target word does *not* occur in the list.

☞ **2.10 Analysis of the efficiency of linear and binary search.** Let us say that executing the **`while`**-block takes w seconds — in practice just a microsecond or less — and to make things easier, let us not make a distinction between the **`while`** in `linear` and `binary`.

(a) Suppose that searching a non-existent word in a given list takes x seconds. Now we make the list twice as long.
   - How much time does a linear search take now?
   - How much time does a binary search take now?

(b) Can you give a mathematical function that expresses how much time searching for a non-existant wordt takes, in terms of the length of the list?
   - For linear search;
   - For binary search.

□

To test the difference in efficiency experimentally, the lab files also include a script `searchmeasure.py`. This script measures the time for linear search versus binary search of a word in microseconds. The measurement is not very accurate, but good enough for our purpose. Call the script from the command line using "`python searchmeasure.py Unabr.dict`.".

☞ **2.11 Experimental observation of the efficiency of linear and binary search.**

(a) Study the script `searchmeasure.py` and make sure you understand what goes on. Run the script and try out a number of input words.
(b) What is the smallest time difference between linear and binary search that you can observe?
(c) What is the largest time difference between linear and binary search that you can observe?

□

2.2.3 Sorting

The above clearly shows that it can be advantageous to work with sorted lists. This, then, makes it important to be able to quickly sort an (initially unsorted) list. In the next exercises you will meet two sorting algorithms.

Algorithm 3: Ordering a list using *bubble sort*

input : List *data*
output: Sorted copy of list *data*

```
1  Assign a copy of data to res ;                          /* res is the result list */
2  Assign the last index in res to ui ;        /* ui is the index of the last unsorted value */
3  while ui > 0 do                                /* If ui equals 0, everything is sorted */
4  |  Assign 0 to si ;                          /* si is the index of the last swapped value */
5  |  Assign 0 to i ;                          /* i is the index that is currently processed */
6  |  while i < ui do                          /* If ui is reached then the rest is ordered */
7  |  |  if res[i] > res[i+1] then                        /* res[i] en res[i+1] are inverted */
8  |  |  |  Swap res[i] and res[i+1] ;
9  |  |  |  Assign i to si ;                              /* We just swapped at index i  */
10 |  |  end
11 |  |  Increment i by one;
12 |  end
13 |  Assign si to ui ;                              /* The list is ordered from si+1 */
14 end
15 return res
```

The first of the two is Algorithm 3. This algorithm is called *bubble sort* to convey the corresponding mental image: larger values in the list "bubble" upwards, towards the end of the list. To understand in more detail what the algorithm does, you can best try it out manually using a (small) example.

☞ **2.12 Manual execution of bubble sort.** What happens if you perform Algorithm 3 on a list *data* with the following content (note that the list elements are now placed next to each other instead of on top of each other):

Index:	*0*	*1*	*2*	*3*
Value:	2	4	1	3

Answer the question by manually going through the steps of the algoritm, and each time drawing the state of the variables. □

If you can answer the following question correctly, you are a long way towards understanding this algorithm well.

☞ **2.13 Analysis of the efficiency of bubble sort.** Consider the performance of the algorithm in the most extreme cases:

(a) How does it perform when applied to a list that is already correctly ordered?
(b) How does it perform when applied to a list that is entirely in reverse order?

In particular, in both of the above scenaris, answer the following questions:

- How often is the **while** in line 3 executed?
- How many times the **if** in line 7?
- How does the time taken by the algorithm depend on the length of the list?

□

To program bubble sort in PYTHON, you need to know a few more things.

- To copy a list (line 1) in PYTHON, say the value of `data`, use `data[:]`; for instance, `result = data[:]` will assign a copy of the list `data` to `result`. The fact that it is a *copy* means that afterwards, `data` is unchanged when `result` is modified, and vice versa.
- To swap two values (line 8) in PYTHON, use so-called *simultaneous assignment*: for instance, `x,y = a,b` has the effect that afterwards `x` has the value of `a` and `y` the value of `b`. This is expecially useful in a construction such as `x,y = y,x` which swaps the values of `x` and `y`.

☞ **2.14** Program a PYTHON module `sort` with a function `bubble` implementing Algorithm 3. Test your implementation in the interpreter, with a few simple lists of numbers but also by sorting the words in the lab files `Unabr.dict` and `hacktest.txt`. (Use the previously discussed function `util.words` for converting these files to lists of words.) Which file takes more time to sort? Can you explain this? □

Bubble sort does not belong to the fastest class of sorting algorithms — that is, unless the list was already sorted to begin with. A smarter sorting algorithm can save as much time as a smarter searching algorithm. Many of the smarter algorithms use *recursion*: the original problem is first solved for a smaller list, and that solution is used within to solve the original problem.

You will now use *merge sort*: this belongs to the category of smerter algorithms. Merge sort is a recursive algorithm based on a *divide-and-conquer* strategy: to sort a list, first sort the first half and the second half, then merge them together into a single whole. The pseudo-code of merge sort is given in Algorithm 4.

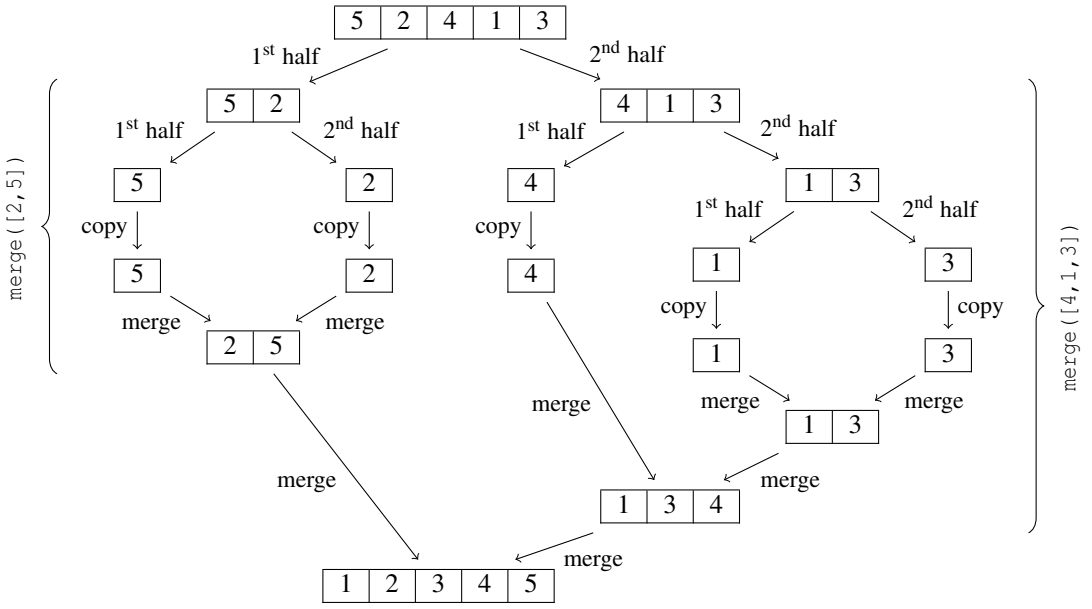
Algorithm 4: Ordering a list using *merge sort*

input : List *data*
output: Sorted copy of list *data*

```
1  if data has 0 or 1 element(s) then
2      return a copy of data
3  else
4      Sort the first half of data and assign the result to fst ;
5      Sort the second half of data and assign the result to snd ;
6      Assign an empty list to res ;                          /* res is the result list */
7      Assign 0 to fi and si ;          /* fi and si are increasing indexes in fst and snd  */
8      while fi and si are valid indexes in fst and snd, respectively do
9          if fst[fi] < snd[si] then                          /* fst[fi] is smaller than snd[si] */
10             Append fst[fi] to res ;
11             Increase fi by one;
12         else                                  /* snd[si] is smaller than (or equal to) fst[fi] */
13             Append snd[si] to res ;
14             Increase si by one;
15         end
16     end
17     if fi is a valid index in fst then
18         Glue the rest of fst (from fi onwards) to the end of res ;  /* All snd-elements have been
           treated */
19     else if si is a valid index in snd then
20         Glue the rest of snd (from si onwards) to the end of res ;  /* All fst-elements have been
           treated */
21     end
22     return res
23 end
```

The recursion appears in lines 4 and 5: this refers to "sort the first half" and "sort the second half". The idea is that *data* is split into two sub-lists of equal length; each of these sub-lists is subsequently sorted through a call of `merge`.

For instance, we can visualise the execution of `merge([5,2,4,1,3])` as follows:



This shows that the list is split each time into "1st half" and "2nd half" (lines 4–5 in Algorithm 4), which then are first sorted themselves (lines 7–21). Convince yourself you understand how the merging works, for instance by merging `[2,5]` and `[1,3,4]` into `[1,2,3,4,5]` step by step, using pen and paper.

☞ **2.15 Manual execution of merge sort** Visualise the execution of `merge([4,3,1,6,2,5])` in the same way as above. □

To program merge sort in PYTHON, you need to know a few more things.

- In lines 4, 5, 18 and 20 you have to take a *fragment* of a list; not the entire list and not just a single element. Use `data[:i]` to copy the fragment from `data` from the beginning *up until* (but not *including*) index `i`, and `data[i:]` for the fragment from (including) `i` up until *and* including the last element. As we have already seen, `data[:]` also produces a copy of the entire list.[3]
- In lines 10 and 13 you have to append an element to a list. Use the standard PYTHON function `data.append(elem)` to append the element `elem` to the end of `data`.
- In lines 18 and 20 you have to glue a list to the end of another (existing) list. Use `data.extend(rest)` to glue the list `rest` to the end of `data`.

To properly understand these expressions, you can always try them out in the PYTHON interpreter.

☞ **2.16** What is the difference between `a.append(b)` and `a.extend(b)`? First think what the answer should be and then try it out, for the cases

(a) `a=[1,2]` and `b=3`
(b) `a=[1,2]` and `b=[3,4]`

□

☞ **2.17** Program merge sort in PYTHON (in the module `sort`) and test your result. (You should be able to show your tests during sign-off.) □

We will now again look at efficiency in practice. As you have already seen from the script `searchmeasure.py` (Question 2.11), you can measure the time a given PYTHON-command takes by using the function `time.process_time()`, in the following way:

```
# import standard module with clock function
import time
```

[3] Expressions of the form `data[i:j]` also exist, but we do not need them here.

```
# measure start and end time when executing command
start = time.process_time(); command; end = time.process_time()
# subtract start from end time; multiply by 10^6 and round to get micros
duration = round((end - start) * 1000000)
```

☞ **2.18 Experimental observation of the efficiency of bubble sort and merge sort.** Compare (by trying out) the execution time of bubble sort and merge sort:

(a) Using `hacktest.txt`
(b) Using `Unabr.dict`

What are your observations? Can you explain them? □

The next question is tricky: at this moment in time you have not yet learned how to answer questions of this kind systematically (this will only be taught in the second year), but maybe you have an intuition for it. Do not spend too much time on this question now, if you get stuck with it; it is optional and does not need to be signed off.

☞ **2.19 Analysis of merge sort (optional).** If a list has n elements, then merge sort takes (in the order of) $n \log_2 n$ steps to sort it — where "a step" may be an assignment or a test in PYTHON. Can you think of an argument why this is the case? □

Through the combination of sorting and searching in a *sorted* list, we can now also more quickly search in an (initially) *unsorted* list: after importing `util`, `ordsearch` and `sort` you can perform the following call:

```
ordsearch.binary(sort.merge(util.words("grail.txt")),"swallow")
```

This sorts the words in `"grail.txt"` en searches the sorted list for the word "swallow".

☞ **2.20 Fast search in an unsorted list.**

(a) The search method explained above is *not* faster than linear search! Explain why not.
(b) If we want to find multiple words in an unsorted list, then it may be faster to first sort the list. From approximately how many target words does this start to be the case, assuming the list contains 10000 words?

□

2.2.4 Duplicates and tuples

Before you can start with the pearl assignment, there are a few more concepts that will be useful.

Firstly: if you sort a realistic text file, it is immediately obvious that a lot of words occur more than once. In some cases you may want all those occurrences to be in the sorted list, but often it is better to keep only one copy of each word. The simplest solution is then to remove the duplicates from the list, simply by creating a new list to which you copy a single instance of each word. Algorithm 5 shows (in pseudo-code) how that can be achieved.

☞ **2.21** Program Algorithm 5 in PYTHON, as function `remove_dups` in a new module `dup`, and test your result. □

☞ **2.22 Analysis of Algorithm 5.**

(a) Write a small piece of PYTHON code that determines the number of different words in `hacktest.txt` and `grail.txt`.[4]
(b) If you use this algorithm on an unsorted list, not all duplicates are removed. (Test this behaviour.) How do you explain this, and how would you describe what does happen?

□

[4] Note that `util.words` considers punctuation to be part of a word, and that uppercase and lowercase letters are considered to be different. For instance, `"you"`, `"You"` and `"you?"` are different words. You may just work with this outcome and do not have to "clean" it.

Algorithm 5: Removing duplicates

input : Sorted list *data*
output: Copy of *data* in which each value occurs exactly once

```
1  Assign an empty list to res;
   ;                                                    /* res is the result list */
2  if data is not empty then                  /* If data is empty, data[0] does not exist */
3  |  Assign data[0] to fresh ;        /* fresh is a fresh value not yet occurring in res  */
4  |  Assign 1 to i;
5  |  while i is a valid index in data do              /* Continue until the end of data  */
6  |  |  if data[i] does not equal fresh then    /* We arrived at the next distinct value */
7  |  |  |  Append fresh to res ;          /* The previous fresh value is added to res  */
8  |  |  |  Assign to fresh the value of data[i];          /* This is the next fresh value */
9  |  |  end
10 |  |  Increment i by one;
11 |  end
12 |  Append fresh to res ;                /* The last fresh value is appended to res  */
13 end
14 return res
```

The second new concept is to work with lists not just containing simple values such as numbers or words, but containing *tuples*. A tuple is itself also a list, with a fixed length. A tuple of two elements is often called a *pair*.[5]

Since for the notion of a list it does not matter what kind of elements are in it, it is straightforward to construct and use lists of lists (or tuples). The only difference is that the order of the elements is now arranged somewhat differently. In case of numbers or words, you can test the ordering with

```
if data[i] < data[i+1]:
    do something
```

but if the elements of `data` are tuples, then you should ask yourself what "<" means.

☞ **2.23** What is the correct order for the following tuples:

```
["a",3]  ["a",2]  ["b",1]  ["ab",10]
```

□

When asked this question, your first response should be: what do you mean by "correct"? What ordering should we use? The most natural answer (but certainly not the only one) is to use the so-called *lexicographical* sorting criterion: $[x_1, y_1]$ is smaller than $[x_2, y_2]$ if either $x_1 < x_2$, or $x_1 = x_2$ and $y_1 < y_2$.[6] If `data[i]` is itself a pair, then the elements of that pair can be addressed by `data[i][0]` and `data[i][1]`. To test the lexicographical order, we therefore have to use:

```
if (  data[i][0] < data[i+1][0]
   or data[i][0] == data[i+1][0] and data[i][1] < data[i+1][1]):
    do something
```

(Note the parentheses around the condition: these are required because there is a newline between the sub-conditions.)

☞ **2.24** Implement a function `sort.merge_pairs` that sorts a list of pairs lexicographically. Test your program, for instance with the list of Question 2.23. □

[5] This is a white lie: those who know PYTHON also know that tuples and lists are actually different in that language — the main difference being that tuples are immutable and are denoted with round rather than square brackats. For the purpose of this pearl, we only want you to use lists.

[6] It so happens that in PYTHON, "<" is actually defined over lists such that it implements the lexicographical order.

When *searching* in a list of pairs, another thing comes into play. Sometimes yu might want to search only on the basis of the first element of the pair — for instance, if the list consists of pairs of student numbers and grades, and you do know your own number but not your grade.

☞ **2.25** Implement a function `ordsearch.binary_pairs` that, given an ordered list of pairs and a value, looks for this value *as first element of a pair* and returns the index of that pair in the list, or `-1` if the value does not occur in the list. For example:

```
>>> x = [["Anne",7],["John",5],["Pete",9]]
>>> ordsearch.binary_pairs(x,"Brenda")
-1
>>> ordsearch.binary_pairs(x,"John")
1
```

□

2.2.5 Pearl assignment: Build a search engine

You have now learned enough to carry out this week's pearl assignment. The assignment is about searching a given word in a set of text files — the basic functionality of search engines such as Google (apart from the fact that the text files have already been collected and do not have to be reaped from web sites all over the world).

An example collection of text files has been included in the material of this week; these are the files with extension `*.txt` on Canvas

The first step is to read in the text files. This step has been pre-defined: the function `util.all_word_pairs()` returns a list of pairs, each consisting of a word from a text file together with the name of that file. Given this list, your task is to efficiently search for a word and return all the files in which it occurs.

☞ **2.26 Basic assignment.** Implement a new module `pearl` containing a function `make_table(pairs)`, which is invoked with as parameter a list such as the one returned by `util.all_word_pairs()`, and returns a new, sorted list — the so-called *search table* — with pairs consisting of:

- As first element a word (de potential search target)
- As second element a list of files in which this word occurs.

Every word should appear in the search table only once, and the corresponding list of files should not contain duplicates. *The order of the files in the list is not important in this basic assignment.* □

For instance, if we start with two files:[7]

- `brian.txt` containing only the words "eunt" (2 times) and "dead" (1 time)
- `grail.txt` containing only the words "dead" (3 times) and "spank" (7 times)

then the result of `pearl.make_table` should be the following list:

```
[ [ "dead", ["brian.txt","grail.txt"] ],
  [ "eunt", ["brian.txt"] ],
  [ "spank", ["grail.txt"] ]
]
```

To test your result, there is a pre-defined PYTHON script `google.py`, which first calls `util.all_word_pairs` and `pearl.make_table` in succession, and then repeatedly asks for a search term, which is looked up in the search table. *Study this script to understand what is being asked: your solution to the basic assignment is only approved if `google.py` is error free.*

☞ **2.27 Bonus assignments.** The following extensions of the basic assignment can earn you additional grade points::

[7] *This is an example!* The files `brian.txt` and `grail.txt` on Canvas contain the text of the two well-known movies and are therefore obviously much more extensive.

(a) +0.5 *point.* Program a variation `pearl.make_counted_table(pairs)` which for every search term does not list the files in which it occurs in random order, but in decreasing order of the number of times the word occurs in the file. That number may itself be part of the list. For instance, the result of `make_counted_table` for the example above would be:

```
[ [ "dead", [["grail.txt", 3], ["brian.txt", 1]] ],
  [ "eunt", [["brian.txt", 2]] ],
  [ "spank", [["grail.txt", 7]] ]
]
```

(b) +0.5 *point.* program a variation `pearl.make_density_table(pairs)` which lists the files in decreasing order of *frequency* — in other words, the number of occurrences divided by the total number of words in the file. The frequency itself may be part of the list. For instance, the result of `make_freq_table` for the example above would be:

```
[ [ "dead", [["brian.txt", 0.333], ["grail.txt", 0.3]] ],
  [ "eunt", [["brian.txt", 0.666]] ],
  [ "spank", [["grail.txt", 0.7]] ]
]
```

Note: this second bonus question is a good deal more tricky than the first, because you have to have some way of recording how many words there are each text file in total!

□

If you already have some expertise in PYTHON or in programming in general: do not hesitate to implement more functionality! For instance, maybe you want to allow searching for more than one target term at the same time.

What should you hand in? On Canvas, submit *a zip file* containing:

- A text file called `README.txt`, describing
  - Whose solution this is: your name and student number (of both students if you did the assignment together)
  - What you implemented: just the basic assignment (Question 2.26) or also the bonus assignments (Question 2.27); in the latter case, which extensions you did precisely.
  - What you did to test your solution; that is, which commands you tried out and what the outcome was. *Give concrete test cases and output, do not just say you ran the script!*
- The PYTHON module `pearl.py` as described in Question 2.26 and optionally 2.27, plus other PYTHON code you produced yourself insofar necessary to make your solution work. *Add comments to your code to make it understandable.*
- (Not required, but desirable.) A PYTHON-script in which your own test commands (see first bullet) have been pre-programmed.

You do *not* have to include the text files distributed with the assignment.

Assessment The main criteria for assessment are summarised below. *Note that the assignment is not signed off by the teaching assistants; they cannot give you the final word on whether your solution passes the criteria, you have to decide that by yourself by a careful reading.*

- A solution not containing the required elements (see above) — for instance, which does not contain the `README` file or does not say anythiong about testing your own solution — will not be approved.
- A solution of which the code is insufficiently documented runs a high risk of not being approved.
- A solution with the right functionality but a wrong complexity (i.e., which runs too slowly because an inefficient algorithm was used for any of the steps) will not be approved; for the bonus assignment(s), an inefficient solution will receive fewer or no points.

- A solution which passes your own tests (according to your `README`) but which turns out to contain errors upon further testing will only be approved if the handed-in code is decently documented and it turns out to be possible to trace and repair the error easily.

In addition, we test the submitted solutions for code duplication, so as to detect cases of fraud (see Section 2.1.4).

> The algorithm Google uses to order the pages containing a given search term is secret. In any case, it is by no means as simple as in this pearl assignment: it is *not* just a question of counting how many times a word occurs! If you are interested, look up the term *page rank* (in Google!) — and look up *keyword stuffing* to understand why word frequencies are not sufficient on their own.
>
> The reason why Google's algorithm is secret, is that there are enormous economic interests involved: every company would like to appear at the top of the screen! Research has shown that only very few people ever browse to the next page after executing a search command. If you would know Google's sorting criterion, you could analyse what is necessary to appear higher on the result page. In fact, improving the findability of web sites is itself booming business by now.
>
> Conclusion: if you really master the concepts of searching and sorting, you have taken an important step in your career!

2.3 Example test questions

Here you will find a set of sample questions of the kind that may appear in the written test; answers are given in Section 2.3.2. Note that this is *not* a sample test: in a real rest, the questions are chosen such that they are divided evenly over the topics of the lab, and can be answered in 60 minutes.[8]

In addition, the following questions from the lab may also appear in a test:

- Question 2.6 on the difference between numbers and text
- Question 2.7 on searching in a dictionary
- Question 2.9 on the precise execution of binary search
- Question 2.10 on the efficiency of linear and binary search
- Question 2.12 on manually executing bubble sort
- Question 2.13 on the efficiency of bubble sort
- Question 2.15 on manually executing merge sort
- Question 2.20 on efficient search in an unsorted list.

An important point about the test is that you will not be asked to produce large pieces of PYTHON code (though some ability to produce small code fragments is expected), but you will be asked to analyse small algorithms in PYTHON that you have not seen before. Also, you are expected to understand how to use lists, and how the various searching and sorting algorithms encountered during the practical work.

You do not have to independently carry out computations such as the ones in Question 2.20 and 2.19; but you should understand the quantitative difference between algorithms needing $\log_2 n$ steps, n steps, $n \log_2 n$ steps or n^2 steps to process a list of n elements.

2.3.1 Questions

1. Suppose you have a PYTHON list **`list`** known to contain 3 elements; for instance `[10,4,-1]`.

   (a) Write a condition in PYTHON-syntax which tests if this list is entirely sorted in reverse (which is indeed the case for the example list above, but for instance not dor `[10,4,7]`).

[8] Sample tests (with solutions) can be found on Canvas.

(b) Write the minimal PYTHON assignments necessary to make **`list`** correctly sorted (from small to large), assuming that it starts out being sorted in reverse (and still contains 3 elements).

2. Explain under which circumstances linear search in an ordered list can be faster than binary search.

Abalyse the following PYTHON code:

```
1  def insertsort(data):
2      # result list, starts out empty
3      result = data[:]
4      # bound is index of first unsorted element
5      grens = 1
6      # continue until the end of the list
7      while grens < len(result):
8          # elem is element we will now sort into place
9          elem = result[bound]
10         # go back from bound
11         index = bound
12         # look for the correct place of elem
13         while index > 0 and elem < result[index-1]:
14             # go bach until correct place for elem is found
15             # shift elements to the right to make place
16             result[index] = result[index-1]
17             index = index - 1
18         # insert elem into the right place
19         result[index] = elem
20         bound = bound + 1
21     return result
```

3. State what happens at the call `insertsort([4,2,3,1])`, by giving the values of `grens`, `elem` and `result` each time line 10 is reached.

4. How many steps does this algorithm need if `data` is a completely unsorted list of n elements: approximately ("in the order of") n, approximately $n \log_2 n$ or approximately n^2 steps? Explain your answer.

At home you have 100 LPs,[9] which you lend out to your friends on a regular basis. When an LP is returned, you put if back unseen. One fine day, you are fed up with having to search for LPs in your unsorted collection.

For the questions below, explain your answers.

5. How many LP sleeves do you have to look at *on the average* to find a specific one, if your collection is unsorted and does contain the one you are looking for?

6. Describe the algorithm you would use to sort your LPs in natural language. As in Question 2.7, formulate your algorithm in small steps, use numbered lines to describe the steps, and when needed use terms such as "go to line x".

7. How many LP sleeves to you have to look at *at the most* to find a specific one, if your collection is sorted and does contain the one you are looking for, and you search as efficiently as you can?

2.3.2 Answers

1. (a) `lijst[0] > lijst[1] and lijst[1] > lijst[2]`
   (b) `lijst[0], lijst[2] = lijst[2], lijst[0]` (gelijktijdige toewijzing)

2. Linear search can be faster if the target value is "small", meaning that it is or should be near the start of the list. In that case, it is found (or observed not to exist) after a few iterations only.

[9] An LP is a round, usually black disk with a spiral groove on both sides, into which sound is encoded mechanically.

We can make this more precise: binary search takes in the order of $\log_2 n$ steps, so linear search is faster if the target value or a larger one is found within the first $\log_2 n$ elements of the list. *This more precise analysis is not required during the test!*

3. The values are in the following table.

bound	elem	data
1	2	[4,2,3,1]
2	3	[2,4,3,1]
3	1	[2,3,4,1]

The final result is [1,2,3,4].

4. The algorithm requires approximately n^2 steps: the outer **`while`**-loop is iterated $n-1$, and the inner loop on the average $b/2$ times where b is the bound, i.e., the number of already sorted elements; b is therefore itself on the average $n/2$. This results in the execution of lines 16 and 17 roughly $(n-1)\frac{n}{4} = \frac{n^2-n}{4}$ times, which is in the order of n^2. *(During the test, it is not expected that the calculation is done as precisely as this, but it is expected that the outer and inner **`while`**-loops are ditinguished and it is realised that the number of steps is the product of the iteration counts of those two loops.)*

5. On the average you have to look at 50.5 LP sleeves, assuming that for every LP the chances that it is borrowed are equally large. Formally, this number is determined by summing up the number of comared sleeves for each LP in the collection and dividing the total by 100:

$$(\sum_{i=1}^{100} i)/100 = \frac{100}{2} \times 101/100 = \frac{101}{2} = 50,5$$

(The exact calculation is not required — that requires knowing the Gauss-formula $\sum_{i=1}^{n} i = \frac{i}{2}(i+1)$ — but the number 50 does not get full marks. The consideration on the distribution of probabilities may be omitted.)

6. There are many ways to do this, but a credible algorithm is::

   (a) Make a pile of sorted LPs
   (b) Take the next unsorted LP
   (c) Look for the position in the pile where this LP belongs. (This can be done using binary search.)
   (d) Put the new LP into its position, causing the pile to be 1 higher
   (e) If there are still unsorted LPs, return to line 6b.

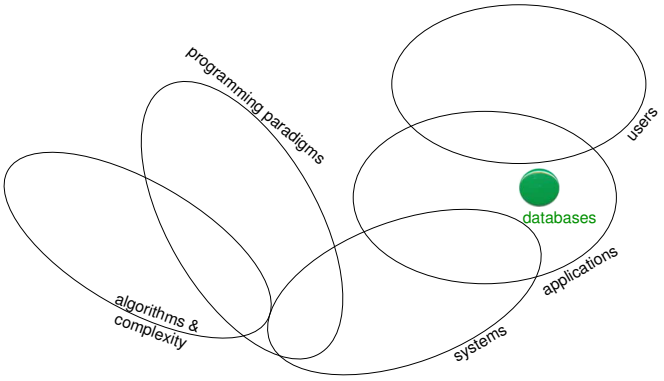
   It is, of course, inadvisable to pile LPs, so after you are done you should put them straight at once!

7. In the worst case, the LP you look for is in (for instance) the second position (index 1). Then the index you will look at (m in the binary search algorithm) is subsequently

   49 24 11 5 2 0 1

   The number of comparisons is then 7. You can reason that it can never be more than 7, since $2^7 = 128 > 100$ (or in other words $7 > \log_2 100$), so after 7 comparisons you have covered all positions. *(Again, the calculation does not need to be done this precisely, but the answer 7 and the connection to powers of 2 should be mentioned.)*

The green pearl seemed to corrupt with greed and selfishness all those who touched it
Jack Vance — The Green Pearl



Week 3

Pearl 010: Databases

3.1 Introduction

The third pearl of computer science we study is "Databases". The perspective we take is the question "How do you effectively and robustly manage large amounts of data that have to be queried, supplemented, corrected, shared and should not be lost or corrupted?". In the pearl we pay attention to data modelling and the managing, querying and manipulation of data in a relational database using SQL. The ambition is not to go in depth, but to be able to tackle a simple scenario.

Module 1.4 "Data and Information" is shared by BIT and TCS, and requires some knowledge beforehand. BIT students are taught basic knowledge on databases in the BIT module 1.3. This includes database schema design with ER-diagrams, database creation and SQL. This pearl offers similar foreknowledge for the "Data and Information" module for TCS-students.

3.1.1 Global description of the pearl assignment

The practical part of the pearl consists of two parts:

- The *SQL-tutorial*. In this tutorial, the database language SQL is practiced. We use a database with information about movies.
- The *Pearl assignment*. In the real world, much information is managed and shared in "office documents". You will get a spreadsheet with information from a student registration system. For conveniency, you get the same data from the spreadsheets' three tabs as three tables in a database. This structure with three tables is in many aspects a bad data structure, your final product will be much better. The assignment is to design a table structure for this data model, to create these tables in your own database, to convert the current data to the new model and to fill the accompanying tables to finally answer some questions about the data and to execute some manipulations on the data.

You work in groups of two on the assignment; the groups are pre-assigned by the module coordinator.

H	Mon	Tue	Wed	Thu	Fri
1	M tst T	P lec T	M self A	P tst T	M self N
2	M tst T	P lec T	M self A	P tst T	M self N
3	M lec T	S tut T	M tut T	M self N	M self N
4	M lec T	S tut T	M tut T	M self N	M self N
6	P lec T	P ass A	P qa T	M tut T	M tst T
7	P lec T	P ass A	P ass T	M tut T	M tst T
8	P prac A	P ass N	P ass N	P ass A	P ass N
9	P prac A	P ass N	P ass N	P ass A	P ass N

Abbr	Form
ass	Assignment
lec	Lecture
pfb	Peer feedback
prac	Practical
pj	Project
qa	Q&A
self	Self study
tst	Test
tut	Tutorial

Abbr.	Part
P	Pearl
F	Final project
S	Academic skills
M	Math

Abbr	Supervision
T	Teacher
A	Teaching assistent
N	None

Mon
- 1–4 Math
- 6–7 The first lecture introduces databases: what is a database, what are they meant for, where they are used, what is special about a database, a bit of history, and a preview of what's to come. Additionally, the most imporant concepts will be explained. Also introduces SQL, the standard computer language for defining, querying and managing data in a relational database.
- 8–9 Lab session SQL; refer to Section 3.2.2. Make sure the assignment is finished within these two lecture hours.

Tue
- 1–2 The second lecture continues with the introduction of databases, specifically data modelling and how to design a table structure for a data model.
- 3–4 Acadamic skills
- 6–9 Start with working on the pearl assignment; a description of the assignment is located in Section 3.2.3. You get a datamodel, and the pearl assignment is to design a table structure for the saved data, in such a way that all the necessary information (including all exceptional cases) can be represented in all information systems and can be used by end-users. We follow the standard *Entity-Relationship* model (ER). You are converting 'old' data to the new structure and answer information-based questions with it.

Wed
- 1–4 Math
- 6–7 There will be an opportunity to ask questions about the theory, the study material, the test, and the pearl assignment. The 'Q&A' takes at maximum 1 lecture hour. The remaining time can be used for preparing for the exam or to work on the pearl assignment. Since we are split over two rooms, the pearl teacher will do the 'Q&A' in SP-4 during the 6th lecture hour and in OH-113 in the 7th lecture hour.
- 8–9 Self study to prepare for the exam and/or work on the pearl assignment.

Thu
- 1–2 Exam
- 3–7 Math
- 8–9 You should finish the pearl assignment during this time, as the rest of the week is mostly math.

Fri
- 1–4 Self study to prepare for the math exam
- 6–7 Math exam
- 8–9 Completing pearl assignment

3.1.2 Study material and tools

The study material for this week can be found on blackboard:

- A general introduction to Databases:
  Modified version of: M. van Keulen, "Gegevensbanken". Chapter VI.4 in Prof.Dr. T.M.A. Bemelmans, Dr.Ir. M. van Keulen, Prof.Dr. R.J. Kusters, Prof.Dr.Ir M. Looijen (eds), "ICT-Zakboek". 3rd edition, 2007. Reed Business. ISBN 978-90-6228-671-3.
- (Multiple SQL tutorials).
- (Documentation on PostgreSQL).

The tools we use:

- PostgreSQL version 9.5.5
- phpPgAdmin version 5.1.

You don't have to install anything. Refer to Section 3.2.1 for details on how to use both.

3.1.3 Mandatory sessions and deliverables

Mandatory elements of this week:

- Signing off Monday's SQL tutorial.
- Participation in the exam.
- Handing in the pearl assignment.

3.2 Description of the sessions and lab assignments

3.2.1 Tools: Database server and front-end

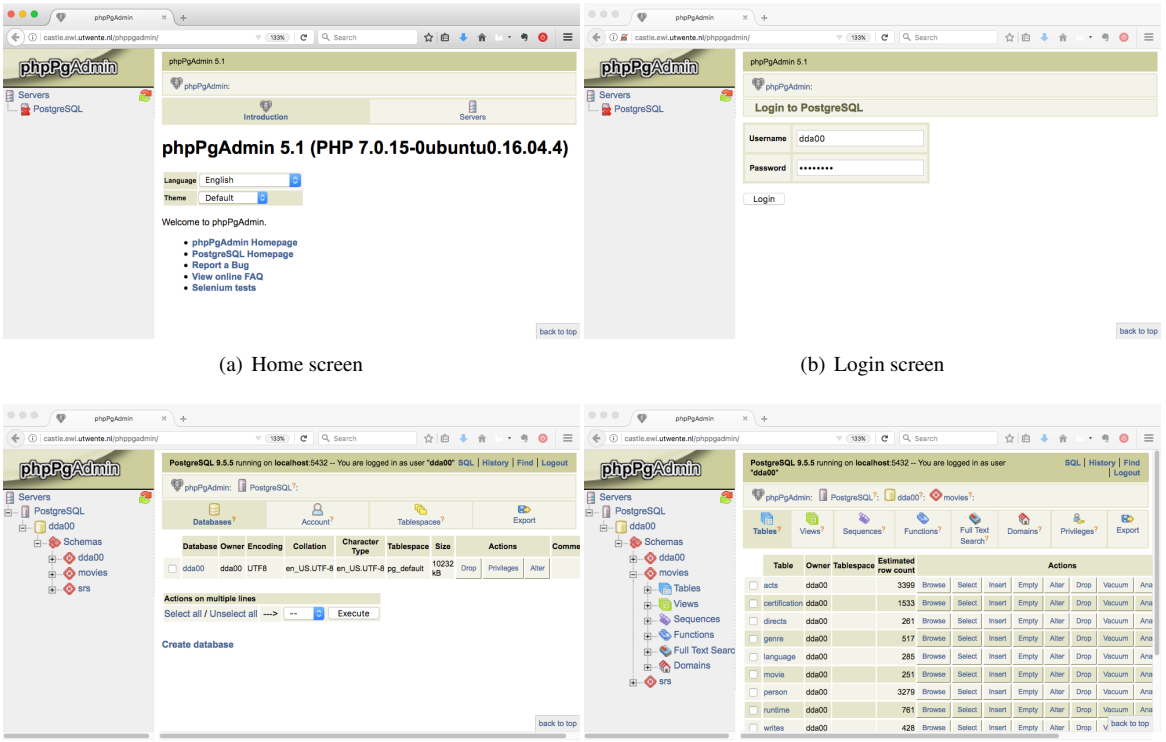
For the SQL practical and the pearl assignment, each group of two students will receive their own database on a database server. You will receive a username and password from the teaching assistants. On Blackboard is a URL to PhpPgAdmin. This is a front-end for the database management system. PhpPgAdmin makes it easier to execute queries and various other tasks on your database. Blackboard also shows what server the database is running on. This is all web-based, so no installation is required.

See Figure 3.1 for some of the screens of PhpPgAdmin. PhpPgAdmin contains more features than you will need, as usual. Below are some instructions on how to reach the two screens that should allow you to finish the entire assignment.
NB: where the screens show "dda*NN*", you will see "db*NN*".

On the home screen, click "PostgreSQL" in the left column to login. After logging in you will see all databases you can access: a database named 'db*NN*' (*NN* being a number), this is *your* database. The database contains three *schemas*, i.e., collections of tables:

- **db***NN*: an empty schema in which you will need to insert and fill the resulting tables of the pearl assignment.
- **movies**: a schema that we use for the SQL practical containing tables with information about movies.
- **srs**: The meaning of the schema is "StudentRegistrySystem"; it contains the same data as the spreadsheets.

If you click on your database or on "Schema" and then the "SQL" tab, you will reach a screen where you can execute arbitrary SQL queries. You can enter SQL queries by copy pasting from a text editor or by running an entire SQL file. Another useful screen is reached by clicking the '+' button to the left of a schema to expand it, then clicking 'Tables'. From this screen you can easily Browse the contents of your tables, Empty them, or Drop (remove) them.



(a) Home screen

(b) Login screen

(c) Logged in

(d) Database 'Test' expanded and 'Tables' clicked

Figure 3.1: phpPgAdmin screens (where the screens show "dda*NN*", you will see "db*NN*").

A good way of doing things is to write all SQL queries you use in an SQL script in a text editor and saving them as one or multiple files. These need to be handed in in the end, but it also allows you to easily empty or remove tables and then creating and filling them again with a few copy paste actions.

If you accidentally mess up the srs or movies tables, these can be restored similarly. On Blackboard you can find a "dump" (SQL file) for both databases. Go to the SQL tab, click "Browse", select the correct dump and execute it. This will restore the tables of the schema.

3.2.2 SQL practical (Monday 8–9)

Introduction

During the first pearl lecture you received an introduction to SQL. It is important to practice SQL well before starting the pearl assignment.

SQL-tutorial

NB: Students who have experience with SQL may choose whether they will follow this tutorial or skip to Question 3.1 "SQL practical exercises".

A good way for getting a first introduction to SQL is to go through an SQL tutorial on the web. There are many good ones. I would recommend W3Schools[1]. From the W3Schools SQL tutorial, you can skip the items with 'Having', 'Exists', 'Any' and 'All' under "SQL Tutorial". Under "SQL Database" you can stop at 'Foreign key'. The other items are not used in this pearl.[2]

[1] `https://www.w3schools.com/sql`

[2] "Not used in this pearl" obviously doesn't mean you should not complete the full tutorial if you're interested.

SQL practical exercises

☞ **3.1** The actual SQL practical consists of writing some SQL queries about movies. As preparation it is a good idea to look through the tables, attributes and data to get an idea of how the database is structured. The names are pretty straightforward. The schema is named 'movies'. Click on the schema and click "Tables" to see all tables. You can view the data in a table with "Browse", and by clicking a table name you can see its attributes.

When you have familiarized yourself with the data, you can start with the exercises below. **Note**: do not forget to make an SQL script containing all solutions as described in Section 3.2.1. The exercises also indicate how many rows the solution should produce. This can be used to check if the solution is roughly correct.

Note: The db*NN* schema is the default schema, but the tables for these exercises are in the "movies" schema. Because of this, table names should be preceded by "`movies.`", for example "`movies.genre`" for the `genre` table.

(a) Return all movies from the year 2000 (8 movies).
(b) Return the name and year of all movies with the genre 'Drama' (143 movies).
(c) Return the name and year of all movies that have both 'Drama' and 'Crime' as their genre (18 movies).
(d) Return the name, role, movie name and year for all actors who have played a role containing 'Tom', sorted by actor name, then year (26 actor-movie combinations).
Tip: you can use '`LIKE`' to do substring matching; just look up how this works.
(e) Return for all directors the amount of movies they have directed, sorted from many to few movies (165 directors).
(f) Return for each language the amount of movies in that language and the amount of actors that played in these movies (20 languages, so 20 rows. For 'Japanese' there are 6 movies and 64 actors. If you get 78 actors, keep in mind that you want the number of *unique* actors).

□

Finalization

Ask a teaching assistant to sign off your SQL script containing the solutions of the SQL practical. If you are not finished by the end of Monday afternoon, show your work to the teaching assistants anyway. They can advise you on what to do: continue for a bit longer or stop working on the exercises and start with the pearl assignment.

3.2.3 Pearl assignment (Tuesday through Friday)

Introduction

In practice, a lot of information is managed and shared through "office documents". You will be analysing how effective this is by working with it yourself through a fictive case. On Blackboard you will find a spreadsheet named 'DBparel-SRS.xls' containing information on a student registry system. This information was received from an 'educational management office'. On Blackboard there is also a second spreadsheet containing the grades of students ('DBparel-SRS-Grades.xls'). These were received from a 'grading administration office'. All information in these files is fictitious. While the names of the courses, students and teachers are real, the rest of the data has been made up.

You can probably think of many reasons why a spreadsheet is not the best option for a data management tool. This does not mean spreadsheets are a bad choice by definition; simply that they are not always a good one either. A carpenter might want to do everything with his trusty hammer, but getting a screw into wood is much easier with a screwdriver.

For this exercise you will design and realize a better system for the student registry system (SRS) based on relational database technology. This will be done in three steps:

1. Design a new database schema by designing and realizing a suitable table structure for a given ER model.
2. Fill the new tables with data.
3. Execute several information processing tasks with the new schema.

The exercise is set up in a way that lets you become more familiar with SQL in addition to the SQL practical, as this will be needed to execute these three steps. The exercise is also split up into two parts: the above three steps are first executed on (a) a simplified version of the system that only considers courses, students and teachers. Afterwards they are executed again on (b) a more complex version of the system that also considers teaching assistants, exams and grades. This second part is optional (bonus exercise).

What to deliver?

What should be handed in on Blackboard for this pearl assignment is:

- Your newly designed table structure (list of tables with for each table the attributes and key(s); this may also be an image).
- *SQL script* that creates and fills your database and executes the information processing tasks.

An SQL script is simply a text file containing all SQL commands in the correct order, meaning they can be executed on an empty database in that order without any errors occurring.

The spreadsheet

The two spreadsheet files contain a total of three sheets.

Education This sheet contains information on what courses are given. It contains the course code, the course name and what study this course primarily belongs to. Each course can occur in multiple rows: one per quarter of a year. For each quarter, the head teacher for that quarter is listed (with teacher id and name).

Course description This sheet provides extra information for the courses. It contains a description of the course as a large piece of text. For ease of use, it lists both the course code and course name.

Grades This sheet is found in the 'DBparel-SRS-Grades.xls' file. Each row contains a student (both student id and name), a course (both course code and name), a quarter and a year.

Note: Course names and descriptions are partly in Dutch and partly in English. Since it doesn't matter for the assignments that you can understand these text fields, we simply left them as is.

Designing a new database schema

To save you some effort, the data from the spreadsheets has been inserted into the 3 tables of the "srs" schema as accurately as possible. The table structure is as follows:

courses course_code, course, description.
For each course, this table contains the course code, the course name and a long description.

education course_code, course, study, teacher_id, teacher, year, quarter.
For each time a course is given, this table contains the course code, course name, quarter and year, what study it belongs to, and the id and name of the head teacher.

grades student_id, student, course_code, course, grade, year, quarter.
For each grade that a student has received, this table contains the student id, student name, course code, course name, quarter and year, and the achieved grade.

This table structure is by far not the best structure for a student registry system. Certain data is duplicated (redundancy) and there are simplifications and incompletenesses (some situations that occur in practice cannot be represented). There are some more deficiencies than the ones shown here.

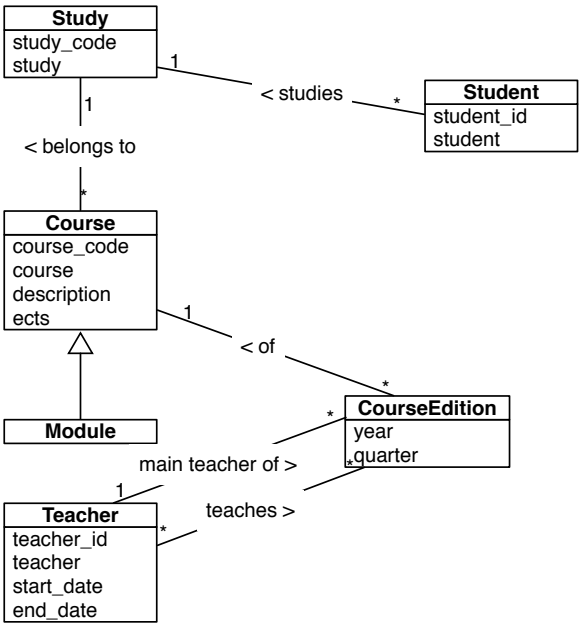


Figure 3.2: New ER model for 'SRS' (only courses, teachers, students and studies)

We will realize a new design for the SRS database. The goal of this new data model is that the student registry system can represent all necessary information (including exceptional cases) for all information systems and end users that need to make use of it. Some requirements for this model are:

- All information needs to be retained, meaning all existing data from the "srs" database needs to be in the new database as well.
- Several useful informational queries and modifications, including exceptional cases, need to be executable (e.g. it needs to know when a teacher has retired).
- It will need to support the assignment of multiple teachers and teaching assistants to a single course.
- With the introduction of the Twente Educational Model (TEM, formerly known as 'TOM'), there are besides 'courses' also 'modules'; these also need to be supported. Modules resemble courses but are larger (more ECTS[3]). Aside from a final grade, a module may also have partial grades for each module part, each with its own name.

We will realize the new SRS system in two steps: first we will focus on courses, teachers, students and studies. After this, we will (as a bonus assignment) focus on teaching assistants, exams and grades. Figure 3.2 contains the first part of the exercise as an ER model. Of course, there can be many variations on this model. These are not necessarily wrong, but may have different pros and cons.

The ER model is structured as follows. Each course has, as mentioned earlier, a course code, a course name and a description. It also has an attribute 'ects'. You may assume that all old courses are 5 ECTS and that modules are 15 ETCS. A 'module' is a special type of course. There is also a 'CourseEdition' entity. This represents one edition of a course, i.e., one occurrence of a course, and contains the year and quarter in which this happened. For example, there is one module "Pearls of Computer Science" that has multiple editions: one for the academic year 2013/2014, one for 2014/2015, and so on. Different teachers may teach in each edition (and teachers can teach in multiple courses). For each edition, there is one main teacher (these are called "module coordinators" for modules). Finally, each course belongs to a study (e.g. "Technical Computer Science") and each student is enrolled in one study (this is a simplification).

[3]From the ECTS, "European Credit Transfer System". One ECTS equals 28 hours of effort. This module gives 15 ECTS. See `http://en.wikipedia.org/wiki/European_Credit_Transfer_and_Accumulation_System`

☞ **3.2** Design a table structure (all tables with their attributes and keys) for the ER model as seen in Figure 3.2. This design needs to be handed in separately as a list of tables with the attributes and key(s) for each table. This may also be handed in as an image.

The `CREATE TABLE` statements that create the tables in your database should be at the start of your SQL script. On phpPgAdmin, under the 'SQL' tab, there is a button labelled "Upload an SQL script" below the text field. This can be used to run your entire SQL script at once, which needs to be able to create your database without errors.

Tip: It is useful to use 'DROP TABLE IF EXISTS' statements before your 'CREATE TABLE' statements. If any of the tables already exist in the database, these will then first be removed. This allows your script to run regardless of the state of your database. □

Filling the new database

Now that we have a better table structure, it can be filled with data. We will begin by transforming the currently existing data. This is also called 'data migration'.

The data from the 'old' student registry system is located in the tables of the "srs" schema. Not all tables, attributes and situations in the new schema will have data available in the old system. Usually, how to fill in the missing data properly is a complex problem. We circumvent this issue here: You can just make up some data of your own.

☞ **3.3** Expand your SQL script with `INSERT` and `UPDATE` statements that take all data from the old tables, transforms it, and inserts it into the new tables. All old data needs to be given a place. Check this by using 'Browse' to see if the number of courses, students, etc. in your new database matches those from the old database.

Tips: Transforming is just a matter of using the correct 'SELECT' queries. The 'INSERT' statement can be used as 'INSERT INTO *table query*'. To avoid duplicate rows, 'DISTINCT' can be used: 'SELECT DISTINCT ...' only produces unique rows. □

☞ **3.4** As said before, not all tables, attributes and situations have data available in the 'old' student registry system. Expand your SQL script with `INSERT` and `UPDATE` statements that fill the empty tables and attributes in the new database with fictional data. Make sure each table contains at least 5 rows. Also make sure that they contain most of the situations that can occur in practice. These are situations such as a module with multiple teachers, a teacher who retired, an internship, a teacher who hasn't done any teaching yet, etc. □

Information manipulation with SQL

To test the new system, we will retrieve and modify some of the information in your new database.

☞ **3.5** Let's start with some basic informational queries. Expand your SQL script with SQL statements for all of these queries. The answer to the first three can be checked by looking at the data in the spreadsheets.

(a) Which students study Biomedical Technology (BMT)?
(b) Of which courses is "van Keulen" a teacher?
(c) Which teachers teach courses about "databases" (meaning courses with the word "databases" in the description)?
(d) For each quarter, how many courses are given?

□

☞ **3.6** A final, important bit of functionality is the modification and manipulation of the data in the database. Expand your SQL script with SQL statements for all of these modifications.

(a) Choose a random teacher. We will assume that this teacher has retired at the end of 2003. Change the data to reflect this.

(b) Courses are often given again in the following year. Make it so that all courses that were given in quarter 4 of 2003 (204 courses) are copied so that these have a 2004 edition as well. Check whether it indeed added 204 rows.
(c) Professor Brinksma has been the rector, hence he won't be giving many courses during that time. In honour of TEM, he still teaches some math lectures, but that is all. Because of this, he was still a teacher in 2003 according to the SRS. He will not be teaching anymore in 2004 though, so edit the resulting data from the previous query so that professor Brinksma's courses are now given by a new teacher: Dr. M. Huisman.[4]

□

Finalization

This is the end of the pearl assignment. This is what needs to be handed in on Blackboard:

- Your newly designed table structure (list of tables with for each table the attributes and key(s); this may also be in an image).
- *SQL script* that creates and fills your database and executes the information processing tasks.

Do you have some time left and want to score bonus points, or would you like to go more in-depth about databases? Try some of the following bonus assignments.

3.2.4 Bonus assignments

The idea behind the bonus assignments is to provide a more in-depth look into the different aspects of the subject. Each of the bonus questions indicates the amount of bonus points that can be scored. You can only get a maximum bonus of 1, so choose the questions that seem most interesting of you. You are allowed to make all of them if you want, but you cannot get more than 1 bonus point in total.

☞ **3.7 Subject**: More complex table design. **Max. Bonus**: 0,5.

Figure 3.3 shows an expansion of the ER model with tests, grades and teaching assistants. The structure is as follows. Each course edition has multiple tests. For a normal course there are two: the test and the resit; for a module there generally are more: tests for the different module parts and their resits. Students can participate in multiple tests and will receive an test grade for each of those (we store both the grade and a name for the grade, e.g. "pearl test databases"). Eventually, these grades together form a final grade for that course for each student. Finally, the teaching assistants. A teaching assistant is basically a student (hence the IS-A relation with student) who acts as a special kind of teacher (hence the IS-A relation with teacher). A student can be a teaching assistant for multiple course editions.

(a) Design a table structure according to the ER model. Make SQL statements that create these tables.
(b) Fill these tables with data from the 'old' SRS database supplemented with made up data for a minimum of 5 teaching assistantships.

Now that we have tables with data on teaching assistants, exams and grades, we can answer information queries about this data.

(a) How many students have failed a course in quarter 2 of 2003?
(b) Make an SQL query that returns an overview of each course with the course name, year/quarter and average grade of that course for that year/quarter.
(c) Make an SQL query that returns a list of all students, with their name and student id, that have been a teaching assistant in a certain quarter (choose a quarter that actually had any teaching assistants according to your data, otherwise the result will be empty).

□

☞ **3.8 Onderwerp**: SQL puzzels. **Max. Bonus**: 0,5.

[4] Module coordinator of the second module; see `http://wwwhome.ewi.utwente.nl/~marieke/`

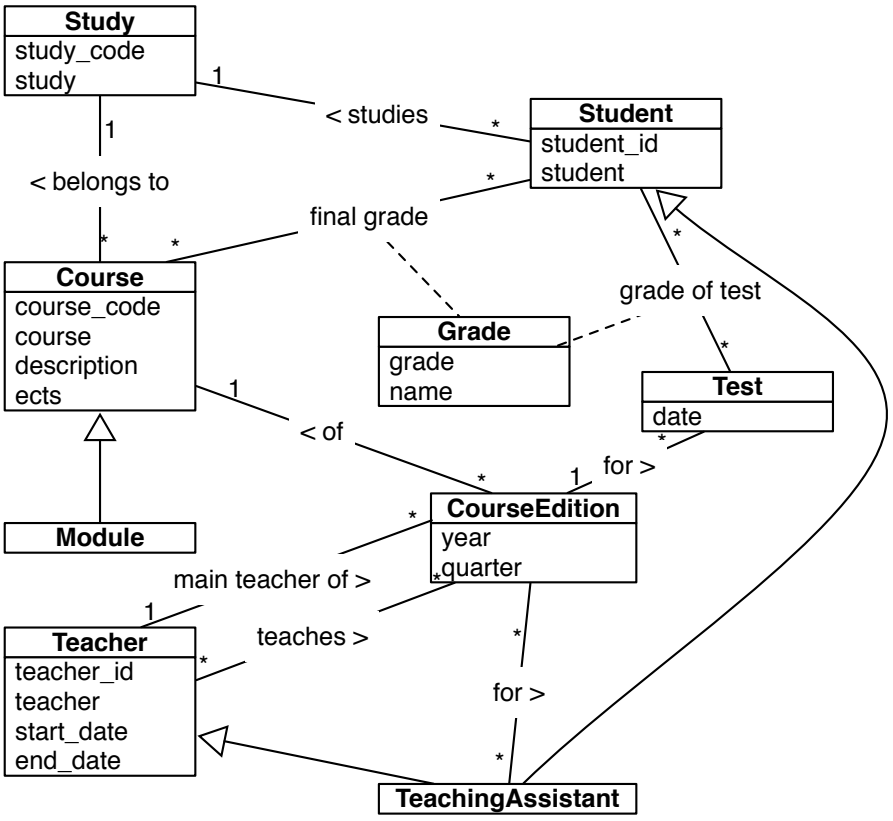


Figure 3.3: New ER model for 'SRS' (with grades and teaching assistants)

For the following "SQL puzzles", the SQL statements we've seen thus far might not be sufficient. Look in the PostgreSQL documentation for the "HAVING" clause of the "SELECT" statement, the possibility of subqueries (a "SELECT" statement nested inside another query), quantifiers such as "EXISTS", and collection operators such as "UNION", "INTERSECT" and "EXCEPT". Only make use of the three tables of the SRS database for these SQL puzzles, so do not use your own table structure.

(a) Make an SQL query that returns all courses (code and name) that have never had a student and that have never been given by a teacher.
(b) Make an SQL query that determines for each student which courses they need to redo (ignore modules here, so determine per student which courses they were graded insufficiently for and no sufficient grade exists).
(c) Make an SQL query that returns, for each year, the course with the lowest average grade and who the teacher for this course is.
(d) Make an SQL query that returns every teacher who, in each year that they have given a course, only gave one course.

□

☞ **3.9 Onderwerp**: Ensuring data integrity. **Max. Bonus**: 0,5.

SQL supplies functionality to ensure data integrity: you can define *constraints* that the database will always have to respect. If any data is changed in a way that violates these constraints, the database will automatically undo (rollback) the changes.

Read the section on constraints in the PostgreSQL documentation: `http://www.postgresql.org/docs/9.5/static/ddl-constraints.html`. Improve your SQL script by adding useful constraints to ensure

Courses	course_code, course, description
	For every course this table gives the course code, name of the course, and a longer description of the course.
Education	course_code, course, study, teacher_id, teacher, year, quarter.
	For each time that a course is given, this table gives the course code, course name, quarter and year the course was given, which study (programme, such as Computer Science) the course belongs to, and the number and name of the main teacher.
Grades	student_id, student, course_code, course, year, quarter, grade.
	For each grade given to a student, this table has the student number and name of the student, the course code and name of the concerning course, the quarter and year of the exam attempt, and the given grade.

Figure 3.4: Description of the table structure of the SRS-database

the integrity of your data. More constraints means safer data. Bonus is determined by how 'safe' you make your database. □

3.3 Example test questions

Apart from exams from previous years, here are some examples of questions that could be asked, with possible answers. In the real exam you will get questions about the same 3 subjects (SQL, database design, databases (general)).

3.3.1 Question 1: SQL (20 points)

We once again use the three tables of the SRS-database from the pearl assignment. The table structure is described in Figure 3.4. The query below returns all the teachers and the courses they give.

```
SELECT DISTINCT teacher, course
FROM education;
```

(a) Explain why the `DISTINCT` is necessary in this query.
(b) Modify the query in such a way that it returns only the teachers who have given a course in 2004.

***Note**: The exam will contain more questions about SQL; as it is worth half of the total points.*

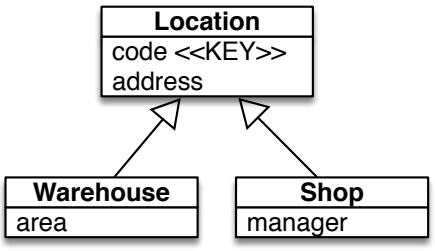
Answers

(a) A teacher can give a course in multiple quarter and years, which could make the query give duplicate results. The 'distinct' makes it so every row is different, i.e. every row is unique.
(b) (every syntactical variation of 'a different study' is accepted)

```
SELECT DISTINCT teacher, course
FROM education
WHERE year <> 2004;
```

3.3.2 Question 2: Database design (10 points)

The data model of an information system of a book store models three entities as below. A database designer has to design a table structure for this ER-model.



(a) Give a possible table structure for this ER-model based on one table per entity. Give your answer in terms of the names of the attribute of these three tables. The types of the attributes may be ignored.
(b) Which tables are used when querying for all adresses of all warehouses? Explain your answer.
(c) Is het possible to store information about these three entities with only one table? Explain your answer.

Answers

(a)
```
Location: code, address, kind --(shop/warehouse)
Store: code, manager
Warehouse: code, area
```

(b) Only 'location' is necessary, as 'kind' could be used to select only the warehouses. If this attribute is not used, both 'location' and 'warehouse' are needed.
(c) Yes. '`Location: code, address, kind, manager, area`', where both manager and area can be NULL.

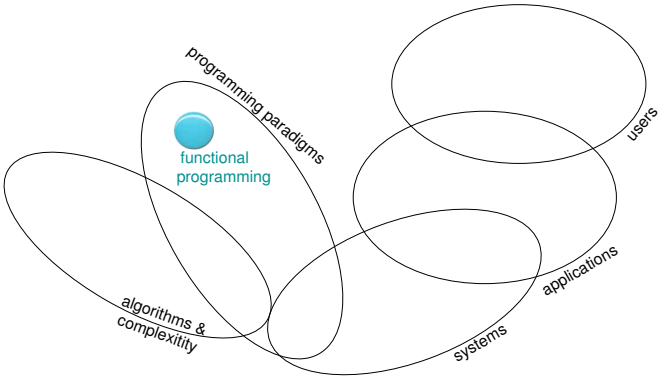
3.3.3 Question 3: Databases (10 points)

(a) Explain what *concurrency* is.
(b) Which ACID attribute or attributes are connected to the problem of *concurrency*?
(c) The ACID attribute *Durability* means that when the DBMS has labeled a finished transaction (COMMIT), the results will not be lost, *whatever happens*. Give two (as distinctly possible) possibilities a database server should be robust against.

Answers

(a) The possibility/fact/problem that multiple users or applications can read/write/manipulate data in the database at the same time.
(b) Isolation: a transaction cannot be influenced by another, or that one transaction cannot 'see' the effect of another transaction while it is not yet finished. Adding Consistency is not wrong: the integrity of the data can be endangered because of concurrent transactions.
(c) Server crash, harddisk failure, network failures, fire, DoS attack.

Dreaming of a bright-eyed girl
Who's got turquoise pearls, seductive eyes
Emblem3 — Tequila Sunrise



Week **4**

Pearl 011: Functional Programming

4.1 Introduction

In Pearl 2 algorithmics was discussed in an *imperative* way, whereas in the current pearl we will approach algorithmics in a *functional* manner. Both ways are called *programming paradigms*. In the imperative paradigm a program consists of a sequence of statements (commands) which prescribe in which way and in what order the state of computer memory should be changed, whereas in the functional paradigm the way in which an algorithm is executed in principle is not the concern of the programmer. In a functional programming language the programmer describes how the result of a program depends on its inputs without specifying which steps should be taken in order to calculate that result. However, sometimes that is unavoidable, for example in case a program otherwise would be too inefficient, but in general a functional programmer only is concerned with the essence of an algorithm. One might say that it's the problem of the computer — and not of the programmer — which steps should be taken and in which order they should be executed. The way in which the programmer writes a functional program is by defining one or more *functions* that specify the result of a program — hence the name "functional programming".

To get a basic intuition of a function one might refer back to the mathematical concept of function, for example:

$$f(x) = 5x^3 + 3x^2 - 6x + 2$$

is a function, in this case a polynome. In the functional programming language used in this pearl — Haskell — one may write this function as follows:

$$f\ x\ =\ 5 * x\hat{}3 + 3 * x\hat{}2 - 6 * x + 2$$

If you now ask Haskell to calculate, for example, $f\ 3$, it will give the answer 146 — which indeed is the correct outcome.

Note that the Haskell formulation of this function is only slightly different from the mathematical formulation. A remarkable difference is that in Haskell the brackets around the arguments of a function are (often) not needed, so instead of $f(x)$ we write $f\ x$.

Even though functions often are mathematical in nature, in practice one usually doesn't think about mathemtics but about the problem for which a solution has to be found. Nevertheless, the implicit mathematical

H	Mon	Tue	Wed	Thu	Fri
1		P lec T		P lec T	P ass A
2		P lec T		P lec T	P ass A
3	P lec T	P ass N	P lec T	P ass N	P ass A
4	P lec T	S tut T	P lec T	P ass N	P ass A
6	P prac A	P prac A	P prac A	P ass A	P tst T
7	P prac A	P prac A	P prac A	P ass A	P tst T
8	P prac A	P prac A	P prac A	S lec T	P self N
9	P prac A	P prac A	P prac A	P ass N	P self N

Abbr	Form
ass	Assignment
lec	Lecture
pfb	Peer feedback
prac	Practical
pj	Project
qa	Q&A
self	Self study
tst	Test
tut	Tutorial

Abbr.	Part
P	Pearl
F	Final project
S	Academic skills
M	Math

Abbr	Supervision
T	Teacher
A	Teaching assistent
N	None

Mon
- 1–2 Free
- 3–4 Lecture FP: basic concepts (functions, application, recursion, typing), introductory examples of function defintions, relation with mathematics.
- 6–9 Practical exercises FP: introductory programming exercises in Haskell.

Tue
- 1–2 Lecture FP: pattern matching, guards, list comprehension.
- 3 Self study
- 4 Academic Skills
- 6–9 Practical exercises FP: continuation of programming exercises in Haskell.

Wed
- 1–2 Free
- 3–4 Lectures FP. For TCS: guest lecture (45 minutes). For BIT: repetition of treated concepts.
- 6–9 Practical exercises FP: finalisation of programming exercises.

Thu
- 1–2 For TCS: Introduction to pearl assignment. For BIT: self study.
- 3–4 Pearl assignment (no assistance present)
- 6–7 Pearl assignment (assistance present)
- 8 Academic skills (guest lecture)
- 9 Pearl assignment (no assistance present)

Fri
- 1–4 Pearl assignment (assistence present)
- 6–7 Test
- 8–9 Finalisation pearl assignment (no assistance present)

Week 4 per session

basis of a functional programming language offers the possibility to prove properties of programs, for example the equivalence of two programs.

In this pearl we will practice the functional approach to programming in a number of examples, among others we will define the sorting algorithms from pearl 2 in a functional way. In addition, we will discuss the relationship with some parts of the lectures on mathematics.

4.1.1 Global description of the pearl assignment

There are two different pearl assignments, one for CS and one for BIT. Both assignments will be published as soon as possible.

The pearl assignment has to be worked out in groups of two.

4.1.2 Study material and tools

As said, in this pearl we will use the functional programming language *Haskell*. On Blackboard (under *Course Materials*) a short introduction to Haskell is available, as well as a list of functions that are predefined in Haskell.

GHCi. We will use GHCi, the standard interactive Haskell Compiler, to be downloaded from `https://www.haskell.org/downloads`. Download the *minimal installer* for your own operating system (Linux, Windows, OS X).

Remark: *Install GHCi on your laptop* before *the first lecture.*

Definitions of Haskell functions are put in a file with extension ".hs". The first line of a Haskell file is as follows:

```
module <Filename> where
```

after which all function definitions follow. Note that the file name starts with a capital letter.

Some hints for using GHCi:

- Start with the command: `ghci`, or with `ghci <filename>`,
- `let <var> = <expr>` gives the value of the expression `expr` to the variabele `var`,
- `:l <filename>` within ghci: loads a file in ghci,
- `:r` within ghci: reload, loads a (possibly changed) file anew,
- `:t <expression>` gives the type of the expression,
- `:e` to edit a loaded file (first do `:set editor <editorname>` to choose an editor),
- `:h` for an overview of the available commands.

Comments in a program are indicated as follows:

"`--`" to turn the rest of the line into a comment,

multi-line comments can be enclosed between "{-" and "-}"

Further information can be found on `haskell.org`, in particular the following pages may be relevant:

`haskell.org/haskellwiki/Tutorials`

`haskell.org/haskellwiki/Books`

There are good online books available:

- *Learn You a Haskell for Great Good*, `learnyouahaskell.com`,
- *Real World Haskell*, `book.realworldhaskell.org`.

Finally, two handy websites are:

- `tryhaskell.org`, an online evaluator for simple expressions,
- `haskell.org/hoogle` to search for various definitions that are available in Haskell.

4.1.3 Mandatory sessions and deliverables

Presence during practical sessions: will be regulated globally over all pearls.

Deliverables. The pearl assignment should be handed in on Blackboard, accompanied with relevant comments.

4.2 Description of the sessions and lab assignments

During the practical sessions a selection of the exercises below has to be made and *signed off* by the assistants. The selection will be published on Blackboard.

☞ **4.1** Introduction exercise. Type in the next function

$$f\,x \quad = \quad 2x^2+3x-5$$

and evaluate it for a few values of x. □

☞ **4.2** Define two functions `circ`, `area` that calculate the circumference and the area (respectively) of a circle, if the radius of the circle is `r` (Haskell knows π as `pi`). □

☞ **4.3** Define two functions `root1` and `root2`, which (given a,b,c) determine the roots of an equations of the form $ax^2+bx+c=0$ (assume that $a \neq 0$). Choose the following outcome in case the discriminant is negative:

```
error "discriminant negative".
```

Write a function `discr` to calculate discriminant and use this function in the functions above.

Test your functions for a number of values of a, b, c. □

☞ **4.4** A second order polynomial is an expression of the form

$$ax^2+bx+c$$

(assume $a \neq 0$).

(a) Write a function `extrX` that calculates the value of x at which the polynomial has its extreme value.
(b) Write a function `extrY` that calculates this extreme value.

□

Intermezzo: Import the module `Char`:

```
import Data.Char
```

Now the following functions are available:

```
ord :: Char -> Int
chr :: Int -> Char
```

which translate a character into its code and back.

☞ **4.5** Write two functions `smallLetter` and `bigLetter` that test whether a character is a small or a big letter (respectively). □

☞ **4.6**

(a) Use `ord` and `char` to define a function `code` that changes a letter (including capital letters) by cyclically shifting it three positions further in the alphabet, i.e., after 'z' one should continue with 'a' again.
For example:
```
code 'a' = 'd'
code 'P' = 'S'
code 'y' = 'b'
```
The function `code` should leave all other characters (digits, spaces, etc.) unchanged. Hint: the relations $<, \leq, \geq, >$ also work for characters.

(b) Evaluate the expressions
```
map code "hello"
map code "Tomorrow evening, 8 o'clock in Amsterdam"
```
(`map` applies the function `code` to all characters in a string).

□

☞ **4.7** Define *recursively* a function `interest` that calculates how much money you have after `n` years, if you start with an amount `a` and receive `p` percent of interest per year. You have to take "interest over interest" into account, where the interest only has to be computed once per year. □

☞ **4.8** Write recursive definitions for the following functions on lists (give the type for every function you define):

(a) `mylength` for the length of a list,
(b) `mysum` that adds the elements in a list of numbers,
(c) `myreverse` that reverses the order of the elements in a list,
(d) `mytake` that gives the first `n` elements of a list (in case `n` is greater than the length of a list, the whole list should be delivered),
(e) `myelem` that determines whether a given element is in a list,
(f) `myconcat` that glues together a list of lists into one long list,
(g) `mymaximum` that yields the maximum of a list of numbers,
(h) `myzip` that transforms two lists into a list of pairs (the shortest of the two lists determines the length of the resulting list).

□

☞ **4.9** Define a function `dividers` that calculates the list of all dividers of a given number *n* (*hint*: use list comprehension). □

☞ **4.10** A prime number is a number that is only dividable by one and by itself. Write a function `isPrime` that tests whether a given number is prime or not. □

☞ **4.11** In how many ways can 100 be expressed as the sum of two prime numbers? In other words, define the list of 2-tuples `(x,y)` such that `x` and `y` are prime, and `x+y=1000`. □

☞ **4.12** A number *n* is a *perfect number* if the sum of all its dividers (except the number *n* itself) equals the number *n*. Examples are 6 (=1+2+3) and 28 (=1+2+4+7+14).

Define a function `perfect` that determines whether a number is perfect. Use your function to calculate all perfect numbers smaller than 1000.

Hint: to remove elements from a list, you may use the operation `\\`. For that you should import `Data.List`:

```
import Data.List
```

Example for the usage of `\\`:

```
[1,2,3,4,5] \\ [3,2]
```

Result:

```
[1,4,5]
```

□

☞ **4.13** A sequence of numbers is *arithmetic* if, starting from some initial number a, every next number in the sequence can be determined from the previous number by adding a fixed difference d.

(a) Write a *recursive* function `r` with three arguments `a`, `d`, and `i` that calculates the `i`'th number of the arithmetic sequence starting with `a`, and using difference `d`.
(b) Let `i` and `j` be two indices. Write a function `total` that calculates the sum of the `i`-th element upto (and including) the `j`-th element.

□

☞ **4.14**

(a) Write a function `allEqual` that determines whether all elements in a list are equal.
(b) Write a function `isAS` that checks whether a sequence is arithmetical (hint: calculate the list of all differences and use the function `allEqual`).

□

☞ **4.15** Write a function that checks whether a list is a palindrome, where a *palindrome* is a list which is the same forwards and backwards. A nice example of a palindrome is (ignore the spaces): "was it a car or a cat I saw"?

Show that your function also works for a list of numbers, e.g., `[1,2,3,4,3,2,1]`. □

☞ **4.16** Write a function that counts how many times a given pattern occurs in a list. For example, the string (the pattern) "abc" occurs three times in the string "babcdecbabcdbabcfe". Occurrences of a pattern may overlap, for example the pattern "aba" occurs two times in the string "ababa".

Remember that a string is just a list of characters. Show that your function works for any type of elements, e.g. for lists of numbers. □

☞ **4.17** A *vector* is a list of numbers. The *dot product* of two vectors is the sum of the products of the corresponding elements of these vectors, for example:

$$[1,2,3] \bullet [4,5,6] = 1*4+2*5+3*6 = 32$$

The mathematical definition of the dot product for arbitrary vectors $\vec{x}, \vec{y}$ (of length n) is

$$\vec{x} \bullet \vec{y} = \sum_{i=1}^{n} \vec{x}_i * \vec{y}_i$$

Define a function `dotprod` that calculates the dot product of two equally long vectors. □

☞ **4.18** A 3-tuple (a,b,c) of integers is called a *pythagorean triple* if $a^2+b^2=c^2$. Well-known examples of such triples are $(3,4,5)$ and $(5,12,13)$.

Write a function `pyth` that generates all pythagorean triples, where a,b,c are all smaller than a given number n. □

☞ **4.19** A list a of numbers is *increasing* if each number in the list is greater than its immediate predecessor. Write a function `incr` that checks whether a list is increasing or not. □

☞ **4.20** Write two variants of a function `contains` that checks whether a list `xs` contains all elements of another list `ys`:

(a) when the order doesn't matter. For example, in this variant the list $[1,2,3,4,5,6]$ contains both lists $[1,3,4]$ and $[3,1,4]$.
(b) when the order should be the same (but not necessarily consecutive). For example, in this variant the list $[1,2,3,4,5,6]$ contains the list $[1,3,4]$, but not the list $[3,1,4]$.

□

☞ **4.21** Write a linear search function `linsearch` that yields the index of a required element `x` in a given list `xs`. If `x` does not occur in the list `xs`, the value `-1` should be yielded.
Hint: first make the list of 2-tuples of the list `xs` and the list of corresponding indexes. □

☞ **4.22** Write a linear search function that yields the list of *all* indexes of the searched element in a given list.
Hint: Use list comprehension. □

☞ **4.23** The *bubble sort* algorithm sorts a list by repeatedly going through the list, while on the way comparing and swapping two consecutive elements if they are in the wrong order. Thus, after one pass through the list the largest element will be "bubbled" to the end. In the next pass through the list, this last element may be ignored, etc. The list is fully sorted if the remaining list to be bubbled does not contain more than one element anymore.

First, write a function `bubble` that implements a single pass through the list, and then write a function `bsort` that implements this process using the function `bubble`. □

☞ **4.24** The *min-max* sort algorithm takes the minimum and the maximum from a list and puts these in front of the list and at the back (respectively). This process is repeated to the list from which the minimum and maximum are removed (use the operation "\\" from `Data.List` for this). Write a function `mmsort` that implements this sorting process. □

☞ **4.25** The algorithm *insertion sort* starts from an empty list and inserts the elements from the given list one after another on the correct position. First write a function `ins` that inserts an element on the correct position in an (already sorted) list. Use this function to write the function `isort` that implements the insertion sort algorithm. □

☞ **4.26** The *merge sort* algorithm splits a list in two halves, sorts both halves (according to the same merge sort way), and combines ("merges") both sorted halves in such a way that the result is sorted again. First write a function `merge` that merges two already sorted lists, and use this function to define `msort` that implements the merge sort algorithm.. □

☞ **4.27** The *quick sort* algorithm divides the list into a list of elements which are smaller or equal to the first element, and into a list of elements that are greater than the first element. This process is the repeated for both resulting lists. Write the function `qsort` that implements this algorithm. □

☞ **4.28** Choose one of the above sorting functions to define a *higher order* sorting function `hoSort` in which the relation by which a list should be sorted is given as an argument `r`. Thus, the definition of `hoSort` has the following form:

```
hoSort r xs = ...
```

and `hoSort` should sort `xs` according to the relation `r`. For example, the following expressions

```
hoSort (<) xs
hoSort (>) xs
```

sort the list `xs` according to the smaller-than and the greater-than relation, respectively.

Note that for a relation `r` it holds that for every two values `x` and `y` the expression

```
r x y
```

yields `True` or `False`. □

☞ **4.29** Define a higher order function `itn` (for "iterate `n` times") with three arguments `f`, `a`, `n`:

```
itn f a n
```

yields the result of applying `n` times the function `f` to the starting value `a`. For example:

```
itn (+2) 0 10  =>  20
```

□

☞ **4.30** Consider the list of 2-tuples that you get by throwing a dice two times, and define functions that yield the following results:

- the list of *all* 2-tuples resulting from two throws,
- the list of 2-tuples of which one of the two values equals `n`,
- the list of 2-tuples which together equal a given value `m`,
- the list of 2-tupels which is sorted by the total value of the two throws.
  Hint: use the function `hoSort`.

□

☞ **4.31** A database contains the following information from persons: name, age, sex, place of birth. You may assume that every person is uniquely determined by his/her name.

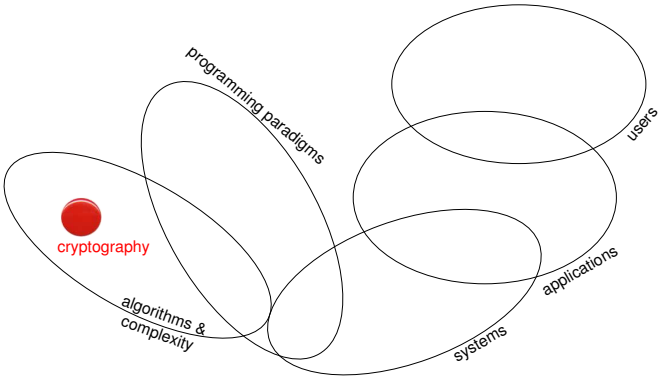
(a) Make an example database (as a list of 4-tuples) to test the functions below.
(b) Write functions to extract the separate data about a person from a 4-tuple,
(c) Write a function in two different ways (with recursion and with list comprehension) to increase the age of each person with `n` years,
(d) Write a function that yields all women from the database between 30 and 40 years,
(e) Write a function that yields the age of a person given by his/her name,
(f) Sort the database by name (*hint*: try the relation smaller-than for two 4-tuples),
(g) Sort the database by age (*hint*: first write a function that "swaps" a 4-tuple to put the age in front, then sort, and then swap each tuple back).

□

4.3 Example test questions

The test will be mostly multiple choice questions and two or three open questions. An example test will be placed on Blackboard. A good way to prepare for the test is making the exercises during the practical sessions.

It is my pleasure to announce the existence of a gem that was unknown until now: the Red Pearl!
Nathan Sternfield — The Secret of the Red Pearl



Week 5

Pearl 100: Cryptography

5.1 Introduction

The field of cryptography is concerned with the design and the analysis of protocols which can cope with (internal and external) adversarial behaviour. It enables mechanisms for secure communications, digital signatures, key exchange, and authentication, to name just a few. In more general terms, it can be seen as the scientific study of techniques for securing digital information, transactions, and distributed computations. In fact, cryptography forms the basis of many important applications, ranging from access control or secure storage to more sophisticated technologies such as secure electronic passports or secure online banking.

In this pearl, we introduce fundamental primitives of the field of cryptography, such as basic encryption techniques and digital signatures. In the previous weeks, you have learned, among other things, how a computer works and how you can program it. The aforementioned fundamental cryptographic primitives will then allow you, for instance, to restrict access to a computer (how can you authenticate?), or to digitally sign a certain piece of software (how can you verify that the latest update to Microsoft Windows that you've installed indeed comes from Microsoft?)

5.1.1 Global description of the pearl assignment

IMPORTANT: You work in groups of two on the pearl's assignment!

The goal of this pearl's assignment is to gain some hands-on experience with encryption and signature techniques. In this context, it is important to understand that these techniques protect against an attacker who is trying to "break" the used techniques. For instance, encryption aims to protect the confidentiality of data, so an attacker will try her best to break confidentiality. Therefore, the pearl assignment contains tasks on the "making" and on the "breaking" of ciphers/signatures.

- In the "making" tasks, you are assigned to properly apply encryption and signature algorithms on given messages. You will have to encrypt short messages with dedicated encryption schemes or sign messages with the RSA signature scheme.
- In the "breaking" tasks, you play the role of an attacker and will break given encrypted messages or forge digital signatures.

H	Mon	Tue	Wed	Thu	Fri
1		P lec T	M self A	P ass A	M self N
2		P lec T	M self A	P ass A	M self N
3	M lec T	S self N	M self N	P self N	P self N
4	M lec T	S self N	M self N	P qa T	P self N
6	P lec T	P ass N	P lec T	M tut T	P tst T
7	P lec T	P ass N	P lec T	M tut T	P tst T
8	P ass A	P ass A	P self N	M self N	P self N
9	P ass A	P ass A	P self N	M self N	P self N

Abbr	Form
ass	Assignment
lec	Lecture
pfb	Peer feedback
prac	Practical
pj	Project
qa	Q&A
self	Self study
tst	Test
tut	Tutorial

Abbr.	Part
P	Pearl
F	Final project
S	Academic skills
M	Math

Abbr	Supervision
T	Teacher
A	Teaching assistent
N	None

Mon 3–4 Mathematics

6–7 Lecture. This first lecture starts with the general organization of the week, gives an overview of the topics and the goals of this week, and discusses the pearl's assignment. After this organizational introduction, we will start with the history of cryptography and some simple (secret-key) encryption mechanisms (such as the Caesar and Viginere cipher, as well as the One-Time-Pad and block ciphers).

8–9 Assignment. Students are assigned to do assignment 5.2 as a preparation for the pearl assignment. Also, they should start working on the pearl assignment by doing assignment 5.6. If there is enough time, students can already do assignments 5.7 (they have all the knowledge to solve these assignments).

Tue 1–2 Lecture. This lecture gives a motivation to one of the most essential primitives in cryptography, called *public-key encryption*. After giving the necessary mathematical background on modular arithmetic, factoring, and the Euclidean algorithm, we will introduce the most famous example of a public-key encryption scheme, called RSA.

3–4 *No Academic Skills session this week; only working on the final academic skills assignment.*

6–7 Assignment. Students continue with assignments 5.6 and 5.7. If there is enough time, students can already do assignments 5.8 or try the bonus assignment 5.10 (they have all the knowledge to solve these assignments).

8–9 Assignment. Do assignments 5.3 and 5.4 and work on the pearl assignments 5.6, 5.7, and 5.8 (see Section 5.2). Some students might want to try the bonus assignment 5.10.

Wed 1–4 Mathematics.

6–7 Lecture. This lecture will give some more details on the RSA cryptosystem and will introduce the RSA digital signature scheme.

8–9 Self study. Students should prepare questions for the Q & A-session on Thursday!
Moreover, students should either work on the pearl assignment (see Section 5.2), recall topics of the lecture, or deepen their knowledge with the study material given in Blackboard (see Section 5.1.2). Students can also start preparing for the short exam/test at the end of the week (see Section 5.3 for examples of assignments/questions).

Thu 1–2 Assignment. Do assignment 5.5 and work on the pearl assignment (see Section 5.2). In particular, students should do assignment 5.9.

3 Self study. Continue working on the pearl assignment and/or prepare questions for the Q & A-session.

4 Q&A. Q & A-session on all covered topics.

6–9 Mathematics.

Fri 1–2 Self study math.

3–4 Self study. Prepare for the short exam/test (see Section 5.3 for examples of test questions).

6–7 Test.

8–9 Self study. Students should finish their work on the pearl assignment (see Section 5.2).

5.1.2 Study material and tools

The main study material for this pearl is this module guide and the lecture slides. Additionally to this, students can benefit from the following study material (to be found on Blackboard):

- Nigel Smart's book "Cryptography Made Simple" is an excellent textbook on cryptography, which covers all the topics we deal with in this pearl. Especially chapter 1 should be highlighted as it gives a very accessible introduction to the necessary mathematical background. The full book can be found here (freely accessible via the UT; and on Blackboard):
  `https://link.springer.com/book/10.1007%2F978-3-319-21936-3`

- For those of you who prefer a less technical (and less complete) introduction to cryptography, we can recommend Ross Anderson's freely available book "Security Engineering". Especially chapter 5 should be highlighted as it gives a general introduction to cryptography that nicely covers the first part of this pearl on the History of Cryptography, Block Ciphers, and Modes of Operation. The full contents of this book can be found here:
  `http://www.cl.cam.ac.uk/~rja14/book.html`

Highly motivated students may also want to look into the freely available "Handbook of Applied Cryptography" by A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, which can be found here:

`http://cacr.uwaterloo.ca/hac/`

5.1.3 Mandatory sessions and deliverables

Required elements in this week are:

- Participation in the pearl test at the end of the week
- Handing-in of the pearl assignment through Blackboard. "Pass/Fail" will also be announced through Blackboard.

NOTE: This is the 5th week of your studies and we would like to stress the importance of you becoming an independent-minded and self-organizing student (these are core goals of the university education). In this pearl, we would like to encourage this by giving you more freedom. Hence, we do not use sign-off lists and we arrange for quite a few slots for self-study. Due to this, it is even more important that you organize your week in a for you workable and efficient way. In fact, *the self-study slots are supposed to be the most learning intensive slots where you recall, reflect on, and internalize the study and lecture materials, continue your work on the assignments, prepare for the next day, and learn for the test at the end of the week.* Due to this, *it is essential that you make the best of the self-study slots!*

5.2 Description of the sessions and lab assignments

5.2.1 Example Assignments that we will do together in the lectures

☞ 5.1 Block Ciphers and Modes of Operation

Important notation. For two integers a and n, we write $(a \mod n)$ to denote the remainder of the division of a by n. More precisely, we can write $a = q \cdot n + r$, where $q := \lfloor \frac{a}{n} \rfloor$ ($\lfloor \cdot \rfloor$ denotes "rounding down to the nearest integer", i.e., cutting off the digits after the comma) and $r := a - q \cdot n$. r is called the remainder of a divided by n and is denoted as $(a \mod n)$.

Consider the following block cipher encryption with $\{0,1,\ldots,7\}$ as the plaintext-, ciphertext-, and key-space (i.e., it has bit-length $n = 3$):

$$E_k(m) = (m+k) \mod 8,$$

where m is the plaintext and k is the secret key used to encrypt messages. Here, the fact that it is a 3-bit block cipher means that it takes 3-bit strings as input (properly converted to their decimal integer representation), then computes the encryption modulo 8, and finally converts the integer output back to a 3-bit binary representation. Moreover, we translate numbers into symbols (and vice versa) in the following way:

0	1	2	3
D	E	G	O

Note that encryption should be space-saving, so if you need to convert a symbol into its binary representation, then use the minimal number of bits (here: 2 bits).

Your task will be to encrypt the message "GOED" with the above block cipher in the ECB-mode by using the secret key $k = 5$.

More precisely, you should finish the following tasks.

(a) Translate the message (a sequence of letters) "GOED" into a sequence of digits as indicated in the above table.
(b) Convert this sequence of digits into its 2-bit binary representation
(c) Divide the resulting bit-string into blocks of bit-length $n = 3$ (think about "why?")
(d) Now that you have the block representation, encrypt this by using the above block cipher in the ECB-mode by using the secret key $k = 5$. Note that you should first convert each block into its decimal representation, then apply the encryption E_k to this decimal representation, and finally convert the resulting ciphertext back to its 3-bit binary representation.

□

5.2.2 Preparative Assignments

Important notation. For two integers a and n, we write $(a \mod n)$ to denote the remainder of the division of a by n. More precisely, we can write $a = q \cdot n + r$, where $q := \lfloor \frac{a}{n} \rfloor$ ($\lfloor \cdot \rfloor$ denotes "rounding down to the nearest integer", i.e., cutting off the digits after the comma) and $r := a - q \cdot n$. r is called the remainder of a divided by n and is denoted as $(a \mod n)$.

☞ **5.2 Block Ciphers and Modes of Operation**

Consider the following block cipher encryption with $\{0, 1, \ldots, 15\}$ as the plaintext-, ciphertext-, and key-space (i.e., it has bit-length $n = 4$):

$$E_k(m) = (m + k) \mod 16,$$

where m is the plaintext and k is the secret key used to encrypt messages. Here, the fact that it is a 4-bit block cipher means that it takes 4-bit strings as input (properly converted to their decimal integer representation), then computes the encryption modulo 16, and finally converts the integer output back to a 4-bit binary representation. Moreover, we translate numbers into symbols (and vice versa) in the following way:

0	1	2	3	4	5	6	7
A	H	C	R	E	I	B	!

Note that encryption should be space-saving, so if you need to convert a symbol into its unique binary representation, then use the minimal number of bits (here: 3 bits).

Your tasks will be to encrypt the message "BREACH" that has been encrypted using the above block cipher in the CBC-mode by using the secret key $k = 14$. You should use the initialization vector IV $(1011)_2$ (binary).

More precisely, you should finish the following tasks.

(a) Translate the message (a sequence of letters) "BREACH" into a sequence of digits as indicated in the above table.
(b) Convert this sequence of digits into its 3-bit binary representation.
(c) Divide the resulting bit-string into blocks of bit-length $n = 4$.

(d) Now that you have the block representation, encrypt this by using the above block cipher in the CBC-mode by using the secret key $k = 14$. Note that you should first convert each block into its decimal representation, then apply the encryption E_k to this decimal representation, and finally convert the resulting ciphertext back to its 4-bit binary representation.

□

☞ 5.3 Greatest Common Divisors

(a) Compute the smallest integer $a > 1$ that is coprime to $b = 20010$ (i.e., such that $\gcd(a,b) = 1$). If you can, try to think of a way to find the smallest integer a without using the Euclidean algorithm at all.
(b) Let $N = 667$ be a product of two distinct primes. Compute Euler's totient function $\varphi(N)$.
Recall that you need to know the factorization of N in order to compute $\varphi(N)$. Therefore, factorize the number N by running through all primes starting with $p = 2$ until you find one with $p \mid N$.

□

☞ 5.4 Modular Arithmetic

(a) Compute $(5^{18370} \cdot 12) \mod 12$.
(b) Compute $(16^{40000} - 35) \mod 17$ (recall that the result has to be ≥ 0).
(c) Compute the inverse element of 13 modulo 396, i.e., the element $0 \leq x < 396$ (i.e., $x \in \mathbb{Z}^*_{396}$) such that

$$13 \cdot x \mod 396 = 1.$$

□

☞ 5.5 Efficient Exponentiation Modulo N

Compute $34^{581} \mod 143$ by using the "modular exponentiation" algorithm. □

5.2.3 Pearl assignment

Important! Always write down the individual steps that you do and why you do them!

Also important! Some assignments can be solved via a trivial exhaustive search (aka brute-forcing). This is not allowed and won't get you any points! For instance, when your task is to break a cipher, it is not allowed to try all possible keys and check whether the decryption yields a proper plaintext. While those attacks exist in the real world, performing them won't make you learn and understand the concepts. Also, you won't be able to perform such exhaustive searches in the test at the end of the week.

The following assignments represent the mandatory pearl assignment. You are required to solve all of them, except for assignment 5.10. If you solve assignment 5.10, you can achieve a bonus point (number between 0 and 1) which, in the best case, equals 10 points in the test at the end of the week, i.e., one grade point in the overall grade.

☞ 5.6 One Time Pad

In this assignment, we consider the One Time Pad with message length n.

(a) In this assignment, we will use the One Time Pad of length n as a block cipher of length n. For messages that are longer than n bits, we will be re-using the key in the One Time Pad and use the ECB mode of operation. Suppose the following ciphertext is an encryption of an English sentence using the One Time Pad with block length $n = 5$ in ECB mode (with "padding"):
`LWPO,KJPM_V?MN,WM,YQP!QM,ZQEQLTXWMVLZQG,WVKTVLJ?LWXQ,X-GI`
Furthermore, assume that decimal numbers are mapped to symbols and vice versa according to the following table:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
_	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P	Q	R	S	T	U	V	W	X	Y	Z	.	,	-	!	?

The mapping from a symbol to a 5-bit binary string can then simply be done by first looking up the decimal number in the table and then converting this number into its binary representation.
Your task is to break the cipher, i.e., figure out the underlying plaintext message and the key that was used in the One Time Pad.

(b) We have learned in the lectures that the One Time Pad is perfectly secure. But why could you break the One Time Pad in (a)? What did we do wrong in the encryption?

□

☞ 5.7 A Block Cipher in OFB-Mode

Consider the following (5-bit) block cipher with $\{0,1,\ldots,31\}$ as the plaintext-, ciphertext-, and key-space:

$$E_k(m) = (m+k)-1 \mod 32,$$

where m is the plaintext and k is the secret key used to encrypt the message. Here, the fact that it is a 5-bit block cipher means that it takes 5-bit strings as input (properly converted to their decimal integer representation), then computes the encryption modulo 32, and finally converts the integer output back to a 5-bit binary representation. Moreover, we translate numbers into symbols (and vice versa) in the following way:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
_	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P	Q	R	S	T	U	V	W	X	Y	Z	.	,	-	!	?

Note that encryption should be space-saving, so if you need to convert a symbol into its binary representation, then use the minimal number of bits (here: 5 bits).

Your task is to encrypt the plaintext "HONEY" with the above block cipher in the OFB-mode by using the secret key $k = 11$, initialization vector IV = $(01001)_2$ (binary), and "shift"-parameter $r = 3$. □

☞ 5.8 Factoring numbers via Euler's totient function

Let $N = pq$ be the product of two secret/unknown primes p and q. Assume that you are given N and Euler's totient function of N, i.e., you learn the value of $\varphi(N)$. So you know N and $\varphi(N)$, but NOT the primes p and q.

Using this knowledge, your task is to find a way to compute the secret primes p and q, i.e., find a formula for p that does not depend on q but only on N, $\varphi(N)$, and known constants.

Hint. First, represent q by the values p, N, and $\varphi(N)$. Use this representation, the fact that $N = pq$, and the "quadratic formula" (high school formula to solve quadratic equations) to compute p. □

☞ 5.9 The RSA Cryptosystem and Textbook RSA Signature

Your task in this assignment is twofold. You will encrypt a given message using the RSA cryptosystem and also sign another given message by using the textbook RSA signature scheme as described in the lecture. For this, consider the RSA-modulus $N = 713 = pq = 23 \cdot 31$.[1]

(a) Compute a very specific RSA public key by finding the smallest public exponent $e > 1$ corresponding to the RSA-modulus $N = 713$ such that $\mathsf{pk} = (N,e)$ is a valid RSA public key (valid = fulfills the properties as described in the RSA key generation algorithm; see lecture slides).
(b) Compute the RSA secret key sk that corresponds to the public key pk from (a). Recall that sk needs to be positive.
(c) Encrypt the message $m = 6$ using the public key pk from (a) with the RSA cryptosystem.
(d) Verify that the signature $s = 17$ is a valid RSA signature for the message $m = 42$ under the in (a) and (b) computed RSA keys.[2]

[1] Please note that this is not a secure RSA-modulus as the primes are way too small! In practice, the primes need to be of at least 1024 bits and satisfy some additional properties (outside the scope of this pearl).

[2] Note that we are using the same RSA public and secret key for the encryption in (c) and the signature in (d). This is only done for educational reasons and is NOT secure in practice.

□

☞ **5.10 (Bonus) The Feistel Cipher**

(a) Consider the Feistel cipher with n rounds ($n \geq 2$) as explained in the lecture; with k-bit key K_{i-1} for round i ($i = 1, \ldots, n$). Let $F : \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ denote the keyed-function in the Feistel design, meaning that it takes a k-bit key K and an l-bit message as input and outputs an l-bit message. Furthermore, denote the n-round Feistel encryption algorithm by $E_{(K_0,\ldots,K_{n-1})}$ and decryption by $D_{(K_0,\ldots,K_{n-1})}$. Show that the Feistel cipher with n rounds ($n \geq 2$) is correct, i.e., show that for plaintexts m of length $2l$ bits, we have:

$$D_{(K_0,\ldots,K_{n-1})}(E_{(K_0,\ldots,K_{n-1})}(m)) = m.$$

(*Hint:* You may want to do an induction on n.)

(b) Implement the Feistel cipher (encryption and decryption) as explained in the lecture, in Python, while considering the following restrictions:

(a) Choose the number of rounds to be $n = 3$. This implies that you need to construct 3 keys K_0, K_1, K_2.

(b) Generate a *random* RSA-modulus $N = p \cdot q$ that has a bit-length of *at least* 256 bits.

(c) Generate three *random* RSA public exponents $e_0, e_1, e_2 \in \mathbb{Z}_\varphi(N)^*$ (i.e., $\gcd(e, \varphi(N)) = 1$ for $i = 0, 1, 2$).

(d) We define $\mathcal{K} := \mathbb{Z}_\varphi(N)^*$ as the key space for our Feistel cipher and take $K_0 := e_0$, $K_1 := e_1$, and $K_2 := e_2$ as round keys.

(e) We define $\mathcal{M} := \{0,1\}^{26}$ as our plaintext/message space.

(f) We define our keyed-function F in the Feistel cipher as:

$$F : \mathcal{K} \times \{0,1\}^{13} \longrightarrow \{0,1\}^{13}, \text{ where } F(K,x) := ((((x)_{10})^K \mod N) \mod 2^{13})_2.$$

Note that $(x)_b$ denotes the representation of x to the base b (e.g., $(x)_2$ means that x is given in binary).[3]

Use your implementation (with the above restrictions) to encrypt the following plaintext message:

$$m = 10011011110010100101011011.$$

After successful encryption, you should decrypt again and verify that the retrieved plaintext message is indeed m again.

Your final implementation/program should output:

- the primes p and q and the RSA-modulus N
- the 3 round keys K_0, K_1, K_2
- the ciphertext of the message m
- the decryption of the encryption of the message m (to verify that encryption and decryption run correctly)

Both the output and the source code of your program are then to be handed in via Blackboard.

Remark: For modular arithmetic (including the efficient exponentiation modulo N), for the Euclidean algorithm, and for generating primes (as well as random numbers), you are allowed to use suitable programming libraries.

□

5.3 Example test questions

Please note that this is not a sample exam! It is just a collection of questions that are similar to those you may find in the final exam. However, please make sure that you also take a look at the example test from 2013 that can be found on Blackboard.

☞ **5.11**

[3] Please be aware that the keyed-function F in a Feistel cipher would never be chosen like this in practice. This is only done for the purpose of this bonus assignment.

(a) What does it mean for a symmetric-key encryption scheme to be perfectly secure?
(b) Give an example of a perfectly secure encryption scheme and explain its encryption procedure.

□

☞ **5.12** Consider the following plaintext message (a 12-bit string)

`101100010110`

Encrypt this message in the ECB-mode by using the following 4-bit block cipher

$$\mathsf{E}_k(b_3b_2b_1b_0) = b_3b_2b_1b_0 \oplus k$$

with the bit-string $k = \texttt{010}$ as secret key (note that $b_3b_2b_1b_0$ denotes an arbitary 4-bit plaintext message). □

☞ **5.13**

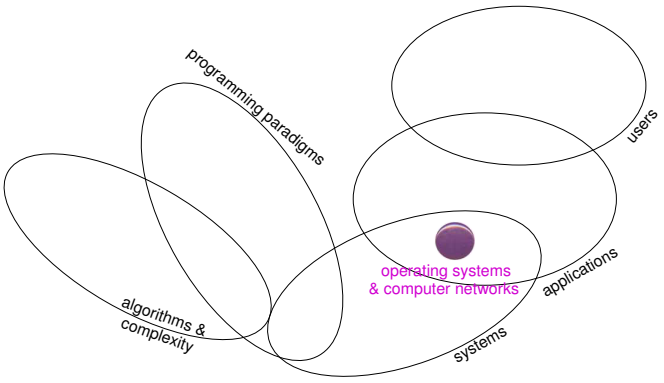
(a) What is the definition of a greatest common divisor of two integers $a, b \in \mathbb{Z}$? And what are the two greatest common divisors of 2 and 4?
(b) Is there a value $x \in \mathbb{Z}$ such that
$$11 \cdot x \equiv 1 \pmod{23}?$$
If so, why? You are not required to compute an explicit solution.
(c) What is the definition of Euler's totient function $\varphi(N)$ for an integer N? And what is the value of $\varphi(8)$?

□

☞ **5.14** Let $(N,e) = (55,23)$ be a public key of the RSA signature scheme. Compute a valid RSA-signature, under the corresponding secret key of (N,e), of the message $m = 46$. If you're using (explicitly or implicitly) the extended Euclidean algorithm or the algorithm for efficient exponentiation, then please make sure that you write down every single step of the respective algorithm. □

☞ **5.15** Let c and c' be encryptions of messages m and m', respectively, using the RSA encryption scheme with public key (N,e). Show that anyone who only knows c, c' and the public key (N,e) (so no knowledge on neither the messages m and m', the prime factorization of N, nor the secret key corresponding to (N,e)) can compute a valid encryption of the product $m \cdot m'$ under the same public key (N,e). Make sure that you verify that your computed ciphertext indeed encrypts the message $m \cdot m'$ by showing that it decrypts to this message using the secret key. □

When she was a little girl
She wore her mother's pearls
Now she wears those purple pearls
Around her neck around the world
RRR Carter — Pearl the Purple Pirate Princess



Week 6

Pearl 101: Computer Networks and Operating Systems

6.1 Introduction

In this pearl, we return to the computer systems themselves. The very first pearl (only for TI students) was about computer systems at the lowest level of abstraction: the actual hardware. We are now going to look at a higher level of abstraction. One thing we then encounter, is the operating system ("besturingssysteem" in Dutch): a piece of software that takes care of all kinds of management tasks in the background, like arranging for several programs to apparently run simultaneously (but actually in quick succession) on the processor. And secondly, we'll have a look at computer networks, of which the Internet is the best known example: computers which are connected together to enable world-wide communication.

6.1.1 Global description of the pearl assignment

In most pearls, you're asked to design and/or build something, usually by programming. However, *research* is also an essential part of a technical study program at the academic/scientific level. That is what we are going to do in the two pearl assignments of this week. We are going to study experimentally how two operating systems behave when running several programs simultaneously. Furthermore, we are going to *observe* how a well-known computer network, namely the Internet, works. You do these assignments in groups of two. In one of the bonus assignments, the programming and the research aspects will meet: it's about writing a computer program to study packet delays by simulation.

H	Mon	Tue	Wed	Thu	Fri
1	M tst T	P lec T	M self A	P ass A	M self N
2	M tst T	P lec T	M self A	P ass A	M self N
3	M lec T	S tut T	M self N	P ass A	P self N
4	M lec T	S tut T	M self N	P ass A	P self N
6	P lec T	P prac A	P tut T	M tut T	P tst T
7	P lec T	P prac A	P tut T	M tut T	P tst T
8	P prac A	P self N	P self N	M self N	P self N
9	P prac A	P self N	P self N	M self N	P self N

Abbr	Form
ass	Assignment
lec	Lecture
pfb	Peer feedback
prac	Practical
pj	Project
qa	Q&A
self	Self study
tst	Test
tut	Tutorial

Abbr.	Part
P	Pearl
F	Final project
S	Academic skills
M	Math

Abbr	Supervision
T	Teacher
A	Teaching assistent
N	None

Mon
- 1–4 Math
- 6 Lecture. In this lecture, you'll get introduced to the purposes and workings of an operating system.
- 7 Movie: "Revolution OS". This movie is about the history of the "Linux" operating system. Linux is based on a totally different philosophy than most commercial software, namely "Open Source" and "Free Software". So in this movie, you'll also learn about that, and about a way of working which is totally different from what you learn elsewhere in this module about requirements engineering.
- 8–9 Start of the operating systems part of the pearl assignment, see Sections 6.2.1 and 6.2.2.

Tue
- 1–2 Lecture about how computer networks work, in particular the Internet.
- 3–4 Academic Skills
- 6–7 Start of the networks part of the pearl assignment, see Section 6.2.3.
- 8–9 Self study: study the theory about operating systems and networks, with a view to tomorrow's tutorial session.

Wed
- 1–4 Math.
- 6–7 Work further on both pearl assignments.
- 8–9 Self study and/or pearl assignments.

Thu
- 1–2 Tutorial. We're going to do calculations about the delay which data encounters on its way through the network. The questions are in Section 6.2.4.
- 3–4 Complete both pearl assignments.
- 6–9 Math.

Fri
- 1–2 Math self study.
- 3–4 Self study in preparation of the test; and a question hour.
- 6–7 Test.
- 8–9 Self study; perhaps finish the bonus assignment(s)?

Week 6 per session

6.1.2 Study material and tools

The study material for this week consists of two parts:

- A document about operating systems, available on Blackboard (based on p. 761–770 from "ICT-Zakboek" (Bemelmans, Van Keulen, Kusters, Looijen, eds.)).
- Several sections from an online textbook about computer networks. This book is available at `http://gaia.cs.umass.edu/kurose/ebook/text/toc.htm`, and the relevant sections are: 1.1, 1.3, 1.4 and 1.5, including all of their sub-sections. Be careful about that: when you click on e.g. section 1.3 in the table of contents, you go to a page which only has sections 1.3 and 1.3.1, but to see sub-sections 1.3.2 and 1.3.3, you have to use the navigation buttons at the top.

The tools we are going to use are:

- A text editor.
- Python.
- Wireshark, which allows you to view data packets on the Internet; you can install it yourself from `http://www.wireshark.org`.
- Traceroute, which allows you to find out what path your data packets take through the Internet; it normally is already on your computer.
- A ready-made Linux installation on a USB-stick which we provide. This allows you to run the Linux operating system on your laptop, without having to install it on your computer's harddisk. But if you already have a "real" Linux install on your laptop, you can of course use that.

Feel free to explore the Ubuntu Linux environment, but here we give a few hints to get you started.

By clicking "Activities" in the top left corner (or hitting the windows key on your keyboard) you can open the menu. We have already put some useful applications in the menu on the left for you (click the bottom entry to see all applications). Of particular importance is the so-called "terminal", which opens a window with a command prompt. Others include gedit (a text editor) and Wireshark.

From the "terminal", you can start e.g. `python3`, or `top` (gives an overview of all running processes; stop it by pressing `q`). You can start the text editor `gedit` either from the menu, or by typing `gedit filename &` in a terminal window.

To connect to the Internet via WiFi, open the drop-down menu at the top right, click wireless and choose a network and select eduroam. Use the following options: Authentication: Tunnelled TLS, anonymous identity: s1234567@utwente.nl (with your own student number, of course), peap version: automatic, inner authentication: PAP, username: s1234567@utwente.nl, password: yourpassword. Now choose the CA certificate. This should be `ADDTrust_External_Root.pem`, this file can be found in the /etc/ssl/certs folder.

6.1.3 Mandatory sessions and deliverables

This week's obligatory elements are:

- Participation in the academic skills session.
- Participation in the test.
- Discussing with a teaching assistant, and signing off, of those sub-assignments where that is indicated.
- Signing off the pearl assignments on operating systems, and doing the OpenEdX assignments on networks.

6.2 Description of the sessions and lab assignments

This week has a set of tutorial exercises (about delays in networks), and two half pearl assignments: one about operating systems and one about networks.

6.2.1 Pearl assignment on operating systems: measuring multitasking

A multitasking operating system must execute different jobs (programs) one after the other. If none of the jobs is waiting for I/O (input/output), this switching back and forth between jobs is done periodically. The goal of this assignment is to study, in multiple operating systems, how this switching back and forth is accomplished, and how much time each of the jobs get.

☞ **6.1** Write a Python program which in a loop continuously increments a variable by one, and whenever this variable reaches say 1 000 000, prints how much time it took to get there, and then starts anew. Basically, the program must repeatedly measure how much time the computer needs to count from 0 to 1 000 000 (but feel free to try other values).
Hint: you can read the computer's clock using the `time()` function from the `time` module. □

This program gives a quick idea of how much CPU power your program gets: the longer it takes to count to 1000 000, the more time the CPU apparently spent on other jobs. You may need to increase that 1000 000 number if your computer is very fast.

☞ **6.2** Use the above program to analyze and compare the scheduler of at least two different operating systems, e.g. Windows (which you probably have on your laptop) and Linux (distributed on a USB stick, see Section 6.1.2), by opening multiple terminal windows[1] and running an instance of your program in each of them.

Note: if you start Python on Linux, by default you get version 2, while you've learnt version 3. To start version 3, type `python3`.

(a) Systematically try first 1, then 2, then 3, and so simultaneous instances of your program, increasing up to at least 8. Make a table, in which for each number of simultaneous instances, you write down what you see, with some detail. (E.g., "all instances take about 10 ms", or "most take 15 ms, but the instance in the 5th window varies between 15 and 56 ms".)
(b) Can you conclude *from these experiments* how many cores your processor has? (Hint: as long as there are fewer programs running than your processor has cores, each program runs on its own core and thus does not need to wait for the others.) And does that match what you know about your computer from other sources, e.g. from what it says under "system information"?
(c) Do you see any difference between the window which is currently selected ("has mouse focus") and the others? Some operating systems give priority to the currently focused window, assuming that the user finds that the most important task.
(d) Do you notice any differences between the two operating systems you compare? □

Discuss your results with a teaching assistant. The TA will look at your data, so please have the table neat and ready when you want to sign off.

6.2.2 Pearl assignment on operating systems: exploring some aspects of the file system

In this assignment, we explore a few aspects of the file system in Unix-like operating systems, such as Linux and MacOS. We do this step by step on the command line. Text in typewriter font (`like this`) are commands that you can type.

☞ **6.3** Make sure you're on a computer running Linux or MacOS. Open a command-line window (terminal). □

[1] On the supplied Linux USB stick, you can open additional terminal windows by right-clicking on the terminal icon in the menu, or by typing `xterm &` in an existing terminal window.

☞ **6.4** Let's first make a new subdirectory (folder) in which we're going to do our experiments: (`mkdir` = MaKe DIRectory)

```
mkdir pearl101
```

and change into it (`cd` = Change Directory):

```
cd pearl101
```

So now our current working directory in this terminal is this pearl101 directory, which is actually somewhat deeper into the hierarchy of the filesystem, as you can see using the `pwd` command (Print Working Directory):

```
pwd
```

□

☞ **6.5** Next, let's make a few text files in this directory, just for testing:

```
echo Hello World > greeting.txt

echo Nice and sunny > weather.txt
```

These commands simply write those words into the named files. To view the contents in a (text) file, use the `cat` command:

```
cat weather.txt
```

To see which files there are in our directory, use the `ls` (LiSt) command:

```
ls
```

If we want to know more about these files, we can add the `-l` (long) option:

```
ls -l
```

This shows us some of the meta-data that the filesystem has stored for each of these files, for example:

```
 -rw-r----- 1 ptdeboer dacs 12 Sep 29 17:29 greeting.txt
 -rw-rw-r-- 1 ptdeboer dacs 15 Sep 29 17:29 weather.txt

  ^^^^^^^^^^ |    |      |    | ^^^^^^^^^^^^ ^^^^^^^^^^^^
permissions |    |      |    |  date&time     filename
            |  owner    |    |
            |           |    size in bytes
       link count    group
```

□

☞ **6.6** As you can see in the above `ls -l` output, on this kind of filesystem each file has an owner and is associated with a group. The permissions tell who is allowed to do what with the file, indicated by letters r, w, and x, meaning 'read', 'write' and 'execute', respectively. This is specified for the owner himself (first 3 characters), then for other members of the same user group (next 3 characters), and for all other users (last 3 characters). So, in the example above, the owner is allowed to read and write both files; group members can read both files but only write to `weather.txt`, and others cannot read `greeting.txt` but can read `weather.txt`. You can modify this with the `chmod` command, like this:

```
chmod o+w greeting.txt
```

The `o+w` part determines what is changed:
The `o` means others; alternatives are `u` for user (owner) and `g` for group)
The `+` means give this permission; alternatively, `-` would remove the permission.
The `w` means write permission; alternatives are `r` (read) and `x` (execute).

(a) Change the permissions of `greeting.txt` such that only others can read it, but only the owner (you yourself) can write it.
(b) Try reading `greeting.txt`. Does the system allow you to do this?
(c) Change the permissions of `weather.txt` such that noone can do anything with it.
(d) Now try to erase `weather.txt` by using the `rm` (ReMove) command:

```
rm weather.txt
```

Can you remove the file, even though you are not allowed to write to it? Isn't this unlogical?

□

☞ **6.7** How much storage space do these tiny files consume? For this, we use the `du` (Disk Usage) command:

```
du -B1
```

This shows how much disk space is used by the contents of the current directory; the `-B1` (that's a digit one, not a letter L) option tells it to present the result in bytes (rather than something else, like kilobytes).

Note that the number printed is much more than the few bytes you put in those files. This is because disk space is allocated in blocks of some fixed size, and such blocks can't be shared among files. Even a very small file occupies an entire block.

(a) Create another small file and see how much the disk usage increases. So what is the blocksize on your filesystem?
(b) Try the same experiment on another file system, e.g. a normal USB stick that you can plug in to your computer, or on a computer running Windows (you don't need to use the commandline commands, you can also look at disk usage statistics somewhere in the file manager or so). Is the block size the same?

Discuss your results with a TA, and have it signed off. □

If you're using Ubuntu Linux, you can do the following fun exercise while waiting for the TA:

☞ **6.8** Install the `sl` software package using the following command:

```
sudo apt-get install sl
```

Explanation:
`sudo` runs the following command using SuperUser privileges.
`apt-get` is the name of the software package management program, Advanced Package Tool.
Lots of software for Linux is available in so-called repositories. This makes it very easy to install and update software.
Now type the following command and enjoy:

```
sl
```

If you want to know more about this `sl` command, you can consult its documentation using the `man` (MANual) command:

```
man sl
```

□

6.2.3 Pearl assignment on networks: experimenting with Wireshark and Traceroute

Please proceed as follows:

- Go to `https://learnintsec.org` , our server running the OpenEdX software.
- Make an account with the same e-mail address as Blackboard uses for you; normally this is your @student.utwente.nl e-mail address, but some may have a different address. Do not use one of your existing passwords, but make a new one, and **write it down**! The system's password reset facility does **not** work!
- Register for the "Pearls of Computer Science" course.
- Go to "Courseware", then to "Lab about networking".
- There are the assignments! Signing-off happens by entering your answers in OpenEdX; you'll get immediate feedback about whether the answers are correct. TAs are around to help you if things are not clear or if you're stuck.
- Note: the OpenEdX system can do way more than we now use; in particular, just ignore the points that can be earned for each question.

Bonus assigments

Two half bonus points can be earned in this pearl.

☞ 6.9 First half bonus point: calculating WiFi efficiency

WiFi and other wireless networks have a radio channel which is shared by multiple computers, each transmitting packets. If two or more computers transmit at the same time, their transmissions will disturb each other, and neither is received correctly.

Let's assume we have 4 computers, and time is divided into equal-length time slots. Furthermore, let's assume each computer at every time slot draws a random number (say, by throwing dice) to decide whether or not it is going to transmit. Then pure chance determines which of the following three possibilities will occur, in each timeslot:

(i) 0 computers transmit: channel is not used;

(ii) 1 computer transmits: data is transmitted over the channel;

(iii) 2 or more computers transmit simultaneously: signals disturb each other, data does not get across.

Only in case (ii) the channel is useful, so you'd want to maximize the chances of that.

Assume that each computer transmits with probability p. What is the probability of (ii) if $p = 0.6$? And how should p be chosen to maximize the probability of a successful transmission, and how efficiently is the channel used in that case? □

Real WiFi has improvements compared to this simple random algorithm, to improve the efficiency.

☞ 6.10 Second half bonus point: simulate the delay in queues

Computing the delay that packets incur in queues ("queueing delay") is not so easy, because the length of those queues varies all the time; that is why we've often neglected this component of the delay in tutorial questions for simplicity. With a dose of mathematics, something can be said about the average queue lengths, but another way of finding out more about them is by using *simulation*. That means imitating the behaviour of the queue in software. In general, simulation is a rather widely usable technique to study how well a system will perform, already in the design phase, before it has been built.

In this second bonus assignment, you are asked to write a simulation program for queues. We assume (for simplicity) that time is divided into slots, in which exactly 1 packet can be tranmitted. Assume there are 4 computers, all sending their packets to 1 router, which has to transmit all of these packets over 1 outgoing link. Assume further that each computer in each timeslot has a 20% probability of transmitting a packet.
Note that this means that on *average* 0.8 packets arrive per timeslot at the router, while it can transmit 1 packet per timeslot; so *on average*, it can keep up, but there may be moments during which a queue does build up.
Note further that just like in the first bonus assignment there are computers transmitting with some probability; the difference is that in the first bonus assignment the packets are lost if multiple computers transmit simultaneously, while in the current assignment the packets are queued in that case.
In Python you can generate random numbers between 0 and 1 using `random.random()`, if preceeded by `import random`.
Simulate 100000 such packets; for each packet, keep track of how long it had to wait and compute the average waiting time.
Also try how this changes if the there is more or less traffic, i.e., if the computers have a higher or lower transmit probability than 20%, and make a graph of the results.
Transmit probabilities exceeding 25% are a special case: what happens then, and why? (Don't just look at your simulation results, since the outcome of your simulation may be misleading; think about what this case means!)
The most interesting case for normal operation of a network is when the transmit probability is between 0 and 25%, so make sure that part is clearly visible in your graph. □

Handing in on Blackboard

Contrary to other pearls, for this pearl only bonus assignments are to be handed in on Blackboard; the rest happens through signing-off (the operating systems part) or through OpenEdX. If you hand in one or both

bonus assignments, please do so as follows:

- In the text box on Blackboard, please write your own name and student number, and those of the person that you worked together with.
- Attach separate files, 1 per bonus assignment, with the following names:
  - For the first bonus assignment: `wireless.pdf` in which you give your calculation.
  - For the second bonus assignment: `queueing.pdf` in which you give explanation, and `queueing.py` containing your simulation program (of course, replace `.py` as appropriate if you used a different language than Python).
- Write name and student number of both students in each PDF file.
- Please do **not** merge the files into a single .zip or .rar (or whatever) archive, because that's a lot more hassle for the grader.

6.2.4 Tutorial exercises about delays in networks

Again, go to the OpenEdX server; this time navigate to "Tutorial about delays in networks".

6.3 Example test questions

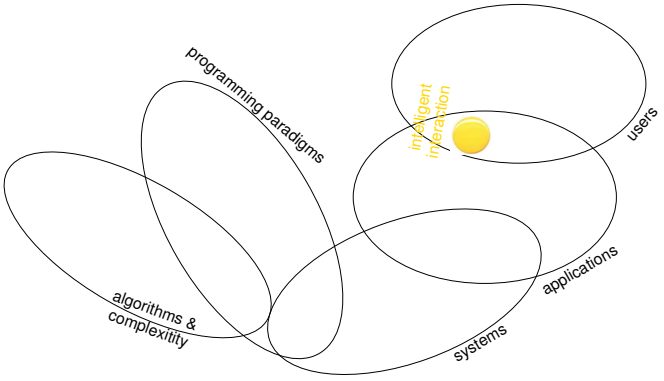
About operating systems:

1. Describe concisely, in your own words, what the function "process management" of an operating system is and does.
2. Under which circumstance does a process go from the state "executing" to "blocked"? Also give an example of this.
3. Is showing a movie a function of the operating system or of an application program? Explain.

About networks:

1. Describe concisely, in your own words, what "protocol layering" is.
2. Theory questions about delay like those in Section 6.2.4.
3. We have seen that TCP uses sequence numbers to number (bytes in) packets and to be able to acknowledge them. Why do we need sequence numbers; can't we just use unnumbered data packets and acknowledgement packets? Explain this by giving an example of how it can go wrong without sequence numbers.
   (Note: this is a question to which you will not simply find the answer in the study material; but you should be able to reason about this yourself, after having worked so much with Wireshark.)

We all must beware of the yellow pearl
Phil Lynott — Yellow Pearl



Week **7**

Pearl 110: Intelligent Interaction

7.1 Introduction

The field of *Artificial Intelligence* tries to understand, to design and to build intelligent entities or artifacts. It is one of the newest research fields in science and engineering. But recently the term Artificial Intelligence is replaced by the term Intelligent Agents. The reason for this is that one cannot design or analyze intelligent agents (artifacts) without taking the environment in which the agent "lives" or operates into account, a so-called situated approach. In one environment an agent could be called intelligent, while the same agent would be called stupid in another environment.

Part of the intelligence of an agent relates to the way the agent interacts with the environment. For instance an advisory agent (for instance a spam filter) could be considered as intelligent if it tags the right email as spam. But what is right? One way to solve this problem is for the agent to observe what the users tag as spam emails. Based on this dataset of spam emails and the dataset of non-spam emails (the emails not tagged as spam) determines a classifier (a software program) which tags the right emails as spam without the intervention of the user. So the decision rules of classifying emails are not determined by the programmer but are extracted from the data. This approach is called Machine Learning.

7.1.1 Machine Learning

A lot of data is stored, but what to do with it? Machine Learning refers to the challenge of making use of data and extract information or knowledge from it. In this chapter we will consider some methods for learning machines to build classification models, such as the spam filter, from data. These models will be used to *predict* properties of similar new data elements or to find otherwise useful *patterns*. Predictive models can either be used for *regression*, in which data is mapped on a real-value (for example the price of a house is predicted based on properties of the house), or for *classification*, in which data is mapped on one of a finite number of class values.

The practical assignment of this pearl (called Pearl assignment) is to build a intelligent spam mail classifier using machine learning. A data set (aka corpus) consists of e-mails that are classified (annotated) as spam or non-spam (non-spam is also called *ham*). The learner builds a model that fits the data. Then the model is used to predict if a new mail is either spam or non-spam ham. A prediction can be right or wrong.

H	Mon	Tue	Wed	Thu	Fri
1	**M tst T**	**P lec T**	**M self A**	P ass N	M self N
2	**M tst T**	**P lec T**	**M self A**	P ass N	M self N
3	**M lec T**	P self N	M self N	**P tut A**	P self N
4	**M lec T**	P self N	M self N	**P tut A**	P self N
6	**P lec T**	**P tut A**	**P lec T**	**M tut T**	**P tst T**
7	**P lec T**	**P tut A**	**P lec T**	**M tut T**	**P tst T**
8	**P tut T**	**P ass A**	P ass N	M self N	P self N
9	**P tut T**	**P ass A**	P ass N	M self N	P self N

Abbr	Form
ass	Assignment
lec	Lecture
pfb	Peer feedback
prac	Practical
pj	Project
qa	Q&A
self	Self study
tst	Test
tut	Tutorial

Abbr.	Part
P	Pearl
F	Final project
S	Academic skills
M	Math

Abbr	Supervision
T	Teacher
A	Teaching assistent
N	None

Mo
1–2 Math test
3–4 Math lecture
6–7 Lecture Intro in to this Pearl and Intro to Bayesian Classification
8–9 Tutorial: Exercises on Bayesian Classification (Chapter 2 handout)

Tu
1–2 Lecture: Baysesian classification and Decision Trees
3–4 Self study: Continu working on the exercises on Bayesian Classfication
6–7 Tutorial: Exercises on Bayesian Classification and Decision Trees (Chapter 2 & 3 handout)
8–9 Pearl Assignment: Weka manual (Chapter 4 handout)
The design and evaluation of a classifiers, Bayesian and Decision Trees, using Weka.
Start working on the Pearl Assignment, see Chapter 5 handout.

We
1–4 Math
6–7 Lecture: Decision Trees and Linear Classification Models
8–9 Pearl assignment (groups of 2)
Self-study. Continue work on the Pearl assignment, including using Decision Trees and Linear Models for Classification.

Th
1–2 Pearl assignment (groups of 2).
3–4 Tutorial: Exercises on linear classification and continue working on Pearl assignment.
6–9 Math

Fr
1–2 Math
3–4 Pearl assignment/Prepare for Exam
6–7 Pearl Exam
8–9 Self study

Performance measures tell how accurate the model is (how many emails are correctly classified, or what the precision or the recall).

Document classification

Classifying emails can be seen as a special case of text or document classification. The bag-of-words model is a popular model used for document classification (documents can be newspaper articles, cooking recipes, e-mails, tweets, blogs, etc.). A bag is a multi-set: elements can occur more than once in the set. A document is modelled by means of a vector where each position in the vector stands for a specific word. In the bag-of-words model the values of the vector elements are numbers that stand for the frequency of the occurrence of the word in the document. Another way to model a document is by means of a *boolean* vector. The value of an element at a specific position in the vector is *true* when the word that corresponds to that position occurs in the document and *false* otherwise.

In this Pearl you will be introduced to a number of classification techniques. First we will start with probabilistic classification and introduce an often used technique in document or text classification which is called "Naive Bayesian Classification". It is based on the classical theory of probability. We will explain the basic principles of this theory. Afterwards we will introduce decision trees and linear classifiers. We will conclude with an introduction in the tool Weka which will used in the pearl assignment and can be used in the final project at the end of this pearl.

The content of this pearl.

1. First introduction to Machine Learning.
2. Introduction into text classification: probabilities, Bayes' rule, bag of words model.
3. Probabilistic classification.
4. Decision Trees
5. Linear Classification Models.
6. How do we measure performance of a classifier.
7. Using the Weka toolkit for Machine Learning

7.1.2 Global description of the pearl assignment

The pearl assignment is to design and validate different models for a spam mail classifier. More information can be found in the reader for this pearl.

7.1.3 Study material and tools

Study materials on Blackboard:

- Reader for this pearl, including exercises.
- Sheets of the lectures.
- Example exam.

Data set and tool used:

- E-mail corpus in arff format (on Blackboard).
- Weka - a toolbox for Machine Learning.

7.1.4 Mandatory sessions and deliverables

Presence is obligatory at:

- Monday 8–9 Tutorial session on Bayesian Classification (Chapter 2 of the reader).
- Tuesday 6–7 Tutorial session on Bayesian Classification, Decision Trees and Linear Classification (Chapter 2 and 3 of the reader).

- Thursday 3–4 Tutorial session Tutorial session on Bayesian Classification, Decision Trees and Linear Classification (Chapter 2 and 3 of the reader).
- Friday 3–4 Pearl Exam

To be handed in at the end of the week via Blackboard:

- Report on the Pearl assignment **in pdf format**.

For more information see the reader of this pearl.

7.2 Description of the sessions and lab assignments

7.2.1 Monday 8–9: Tutorial on Bayesian classfication

In this tutorial session you will start with the exercises on Bayesian Classification, Chapter 2 reader.

7.2.2 Tuesday 5–6: Tutorial on Bayesian classification

In this tutorial you continue working on the exercises in Chapter 2 and start with the exercises in Chapter 3 of the reader.

7.2.3 Tuesday 7–8: Pearl assignment. Weka

Install Weka, read the Weka manual and do the exercises, see Chapter 4 of the reader. Afterwards use the Weka toolkit to train and test the Naive Bayes classifier and Decision Trees for spam emails.
Topics:

- Data visualisation.
- Model construction.
- Naive Bayesian classifier.

7.2.4 Wednesday 6–7: Alternative classification models

First read the chapter on Machine Learning and do the exercises. After that, you can apply Weka decision tree and linear classification (logistic regression in Weka) on the email dataset. You can try out other features like bigrams of words instead of just unigrams of words. More information about text classification can be found on the following YouTube video Text classification using Weka. Since the data is already provided in arff format you can start the video at 6:30 min.
You can work on the final report about spam mail filtering.

7.2.5 Wednesday 8–9: Pearl assignment

Weka is known by now as well as the naive Bayesian method for classification. Two alternative methods for classification need to be applied on the same data. Results in terms of performance measures are compared and reported.

- Training and validating two classifiers; a linear classifier (Logistic Regression) and a decision tree.

  Be aware the the training process can take a lot of time on your computer. When using cross-validation use a small number of folds (5 or even less) instead of the default of 10.

- Performance Analysis. Analyse the performance of the trained models with the confusion matrices.

Report your findings when carrying out the following steps; **describe in max one A4 per subtask**.

- Feature selection
- Model selection: Naive Bayes, linear models, decision trees. Of course, you may test other models available in the toolkit as well.
- Performance analysis and a comparison of the different models.
- Conclusions.
  In the conclusion section of your report you describe what your opinion the best model is and motivate why.

To be handed in

A report in pdf-format. See for content previous section.

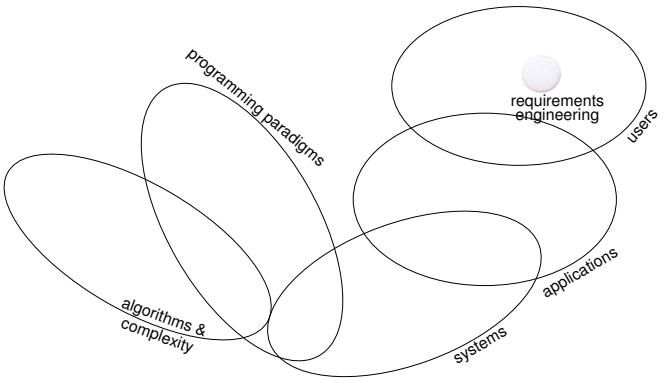
7.2.6 Thursday 1–4: Project: wrap up

During these project hours you can finish the exercises in the Machine Learning reader, your experiments with Weka and the report for the Pearl assignment and Bonus assignment/challenge.

7.3 Example test questions

One can find an example pearl test on blackboard. Answers to this test will be made available on Thursday at 15:00 hour.

All on the board the White Pearl have died
Sonata Arctica — White Pearl, Black Oceans



Week **8**

Pearl 111: Requirements Engineering

8.1 Introduction

Developing software is more than just programming. What should you do to realize a system that makes the customers and the users happy? How do you make sure that the development costs stay within budget? How do you organize a group of people who collaborate to develop a single system? *System development* is a subject that will return in many Computer Science modules. In this pearl we get acquainted with two specific aspects of system development: *Project management* and *Requirements Engineering*.

Projects come in different shapes and sizes. Managing a project that involves a hundred people is a lot more complicated than managing a project that you do with a single team of six. But even for small projects, if you want to make sure that you deliver the right results at the right moment, things need to be managed. In this module we get to know the essential basics of Project Management. (In Module 2, Software Systems, and Module 4, Data & Information, which build on this foundation, you will learn more elaborate management techniques with a specific focus on software development.)

Requirements Engineering is concerned with gathering, specifying, ordering, and maintaining collections of (software) requirements. In this pearl we learn some basich techniques to compose a *Requirements Specification*. A requirements specification is a document that contains the requirements for a system to be developed—not just a list of requirements, but also some context and motivation as to why these are the proper requirements. In the ideal case a requirement specification is clear enough for the client (the person who commissioned the system) to understand what has been specified, as well as detailed enough for the developer to understand exactly what has to be built.

The pearl assignment consists of making a requirements specification for the module project in week 9. This kills three birds with one stone: (1) you can exercise project management on a mini-project (the pearl assigment), (2) you get some experience in gathering requirements and writing a requirements specification, and (3) the requirements specification gives you a head start into next week's project.

8.1.1 Global description of the pearl assignment

The pearl assignment consists of making a requirements specification for the application that you will develop in the module project. You do this with a group of (approximately) six persons.

H	Mon	Tue	Wed	Thu	Fri
1	**M tst T**	**P lec T**	**M self A**	**P tst T**	P ass N
2	**M tst T**	**P lec T**	**M self A**	**P tst T**	P ass N
3	**M lec T**	**P prac A**	**M tut T**	**P ass A**	M self N
4	**M lec T**	**P prac A**	**M tut T**	**P ass A**	M self N
6	**P lec T**	**F lec T**	**P ass A**	**M tut T**	**M tst T**
7	**P lec T**	P ass N	**P ass A**	**M tut T**	**M tst T**
8	P ass N	P ass N	P self N	M self N	**M tst T**
9	P ass N	P ass N	P self N	M self N	P ass N

Abbr	Form
ass	Assignment
lec	Lecture
pfb	Peer feedback
prac	Practical
pj	Project
qa	Q&A
self	Self study
tst	Test
tut	Tutorial

Abbr.	Part
P	Pearl
F	Final project
S	Academic skills
M	Math

Abbr	Supervision
T	Teacher
A	Teaching assistent
N	None

Mon
- 1–4 Mathematics
- 6–7 Lecture Project Management and Requirements Engineering: What is it? Why is it important? What will we do with it this week?
- 8–9 Pearl assignment (Starting phase) (in groups of approximately 6 persons) Getting to know your project group, starting up the project

Tue
- 1–2 Lecture: first steps towards a Requirements Specification
- 3–4 Practical (groups of 2): completing the tutorial assignment from the lecture
- 6 Tool instruction for the module project
- 7 Pearl assignment (Preparation phase): making a planning for the rest of this week, making a task board
- 8–9 Pearl assignment (Execution phase, start)

Wed
- 1–4 Mathematics
- 6–7 Pearl assignment (Excecution phase, contd.)
- 8–9 Self study, preparing the pearl test

Thu
- 1–2 Test
- 3–4 Pearl assignment (Execution phase, contd.)
- 6–9 Mathematics

Fri
- 1–2 Pearl assignment (Closing phase): finishing and delivering a Requirement Specification Document
- 3–4 Self study, preparing the Mathematics test
- 6–8 Mathematics test
- 9 [If needed:] Pearl assignment (Closing phase, contd.)

Week 8 per session

8.1.2 Study material and tools

The following study materials are made available on Canvas:

- *An Introduction to Project Management* (with separate appendix): How to split a project into phases and what to do to guarantee a good result?
- *A primer on Requirements Analysis* (with separate appendices): A small compendium of techniques that can be used to gather requirements and to write them down in a professional requirements specification.

For the pearl assignment we use trello.com, a tool for maintaining taks boards.
See docs.ia.utwente.nl/trello for a brief manual.

8.1.3 Mandatory sessions and deliverables

Participation in the following sessions is obligatory

- Mon 8+9 Pearl assignment – Start of module project (as well as the pearl assignment) with your (brand new) project group!
- di 1+2 Lecture – Includes first bits of the lab assignment
- di 3+4 Practical – The remainder of the lab assignment
- do 1+2 Pearl test
- vr 6–8 Mathematics test

Deliverables for this pearl are the pearl test (see 8.3) and the pearl assignment (see 8.2.4).

8.2 Description of the sessions and lab assignments

In this pearl we make a start with the module project of week 9, by making a requirements specification for the module project. This requirements specification is the pearl assignment for this week.

8.2.1 Mon 8+9: Pearl assignment (starting phase)

These two hours comprise the *starting phase*[1] of the module project, as well as the starting phase of the pearl assignment. For more information about the module project, see Section 0.5 (you find Section 0 on Canvas under *Module Overview*).

Before you start with the assignment proper, you may want to get acquainted with your new team mates—an important part of the starting phase of a project!

For the module project you can choose between two different assignments:

- The standard project is concerned with text mining on a twitter database. To specific stakeholders you present information that is useful to them at the given time in the given circumstances.
- Alternatively, for groups who want a more technical challenge, there is a project about electronic tags for secure access.

Once you know which kind of project you want to do, three tasks remain for this session.

1. The first things that you have to decide as a project group is which kind of application you want to make, and (for the standard project, not the security project) which particular stakeholder you will serve.

[1] See *An Introduction to Project Management* on Canvas

- If you have chosen a particular application it could be useful to draw up a list of all stakeholders you can think of. What could be important for the choice of a stakeholder is wether you have such people among you family or friends. One of the tasks for this week is to interview a stakeholder (see 2.)

2. For the pearl assignment it is obligatory (for all projects, also the security tags) that you interview a stakeholder and submit a report about the interview. That makes it desirable to plan ***now*** how you can arrange for an interview. If there is a chance that the desired interview cannot be realized, you may also want to think of a back-up plan with an interviewee of your second choice.

   - Preferably, you can interview someone you know. It is *not allowed* to call the police, the hospital, ..., to inquire whether you can interview someone. If you really cannot find a stakeholder through friend and relatives, you can ask the teacher to mediate. Possibly he can introduce you to a suitable person.
   - Ideally, the interview should be conducted Tuesday evening, Wednesday afternoon or Wednesday evening. Thursday after the test is bit late—but doable if there is no alternative.
   - Please try to contact an interviewee today, or tomorrow at the very latest. If you have agreed on a time, you can make the planning for the rest of the week tomorrow afternoon.

3. The remainder of today (Monday) you can spend searching for some context information about the application that you want to develop[2]. Try to look at the world from the perspective of 'your' stakeholder.

8.2.2 Tue 3+4: Laboratory session

For the major part of this pearl you will work in project groups of (around) 6 persons. For this lab session, however, please split up your project team and do the exercises in pairs as usual.

During this session you will complete (parts of) the requirement specification[3] that we started to work on during the lecture. Please use the case description below about an app for NS, the major rail transport company in the Netherlands.

- Complete the mission statement.
- Think of some questions you want to ask to a representative of NS Passenger Services.
- Ask these questions when you have the opportunity to do so.
  A representative of NS Passenger Services will visit your tutorial group ***either from 11:30 to 11:45 or from 11:55 to 12:10***.
- Describe relevant requirements as user stories; add acceptance criteria were these are applicable.
- Prioritize the user stories.
- ***Get the assignment signed off by the student assistant before your leave.***

Improving customer satisfaction with NS
NS (*Nederlandse Spoorwegen*) provides rail transport on all major an many of the minor railway lines in the Netherlands. Every day it carries over a million passengers, and the number is going up. However, NS has a problem with customer satisfaction. Many people take the train because it is the fastest, or the only feasible way to get to their destination, not because they like travelling with NS.

The service of NS is in fact better than its reputation. Most trains run smoothly most of the time—90 % arrives in time—but those cases where the service broke down will stay in people's minds. On top of that, there have been some major problems in the last few years, which have dented NS's reputation. The high-speed connection to Brussels has been plagued by major setbacks and political scandals for over a decade,

[2] See *A Primer on Requirements Analysis* on Blackboard.

[3] What is asked here are the sections *3, Mission statement*, and *5, Functional requirements* from the template for a complete requirements specification. In this morning's lecture we covered the *goal* (which is part of the mission statement and would eventually also be mentioned in Section 1, Introduction) and the *stakeholders* (Section 2).

and still isn't functioning properly. In 2016, NS suffered from a bottleneck in train capacity; new trains to cope with the growing demand were ordered but took a long time to get delivered.

NS Passenger Services, the division of NS that is responsible for passenger transportation, is keen to improve customer satisfaction. Some issues, like infrastructure breakdowns, are beyond reach of NS Passenger Services, but there is a lot of things that they can directly influence. One of these is communication to the passengers in case of a breakdown, another is the service quality in the train.

An idea that deserves further investigation is the following: an app, with which passengers can give feedback on their experiences in the train[4]. Or, rather than creating a new, separate app for it, it could be implemented as additional functionality in the existing *Rail planner* app, which the large majority of the regular train travellers have already installed on their smartphone.

The advantages of such a feedback function are obvious. If the windows are dirty, or if there is no more water in the toilet, something can be done about it as soon as the train reaches its destination. Passengers' messages can help to direct service crews to desired repairs and cleaning. Furthermore, if passengers are annoyed about communication in case of delays and disturbances, it can be analysed what exactly went wrong, so that it can be improved in future. But perhaps passengers also want to pay a compliment to conductors for the way they acted in a particular situation.

Improving the services themselves would be the first, but not the only objective of this new functionality. In addition, it would get customers more involved with NS. If customers have the feeling that their feedback is acted upon, their involvment with and regard for NS will increase. Research on customer loyalty has shown that customers who file a complaint will become very loyal customers when the handling of the complaint surpasses the customer's expectations.

This implies that NS Passenger Services would have to handle the feedback appropriately—and possibly inform the senders what has been done with their input. Perhaps text mining can be used to sort incoming messages in those that can be handled with an automatic notification and messages that are important enough to warrant special attention and a personal reply from an NS employee. But these are ideas for the future.

Wil de Jong is head of IT of NS Passenger Services. Wil is very interested to investigate the feasibility of such a feedback function, and has contacted *MR Research* to conduct a pilot. MR research will build a prototype and will investigate whether passengers would like it. The prototype functionality will be offered in a separate app; if the pilot will be successful, the functionality then can be added to the regular NS app, making it available to all customers.

Test persons (passengers who participate in the pilot) should be able to give various kinds of messages. These messages can be read by MR Research and NS Passenger Services, they are *not* passed on to the service staff who would have to act on it if the feedback system would be adopted and the functionality would be made publicly available.

Currently it is not exactly clear what the functionality of the prototype should be. NS Passenger Services and MR Research request your help with establishing the requirements.
More precisely: 'Project 1' is building the prototyp app. 'Project 2' will be the pilot in which this prototype app is used by MR Research to test passengers' reactions. When these are positive, further projects could follow in order to prepare the organization for acting on the feedback and to make the functionality generally available.
Your task in this exercise is to draft (parts of) a requirements specification for 'Project 1'.

8.2.3 Tue 7: Pearl assignment (preparation phase)

The first part of this afternoon comprises the *preparation phase* of the pearl assignment. Make a planning for the rest of the week with your project group. Make the planning in the form of a Trello task board.

1. Find out how Trello works and make a task board with the project group as members. For information about Trello, see docs.ia.utwente.nl/trello.

[4]The idea for this app is partly real, partly fiction. Two UT students have participated in research about the feasibility of such an app, but so far *NS Passenger services* has no concrete plans to implement it.

2. Find out
   - what you need to have finished by the end of the week;
   - which tasks you have to do in order to finish everything.

   For finding out what you need to do, you can use
   - this chapter in the module guide, notably Section 8.2.4,
   - the documents about project management and requirements engineering on Blackboard (Course materials > Pearl 111).
3. When you make a planning, for each task you add a card in the column *To Do*. On the card you can add further details as you like (checklist, who will do it, when should it be finished). Some tasks will be done by the group as a whole (e.g. brainstorm), other tasks can be delegated to subgroups or individuals.

The planning is not a deliverable that you have to hand in—the purpose of making a planning is to get yourselves organized. Nevertheless, on Wednesday afternoon the student assistant may want to have a look at it, just to see whether you are doing all right.

When you completed the planning, you can start the execution. Move the cards for the tasks you will start with from *To Do* to *Doing*.

8.2.4 Tue 8+9, Wed 6+7, Thu 3+4: Pearl assignment (execution phase)

These three blocks comprise the *execution phase* of the pearl assignment. This should include writing a complete draft version of the requirements specification. (The closing phase is for last corrections and making sure that you submit everything as requested.)

Assistance is available Wednesday 6+7 and Thursday 3+4. However, if you run into serious trouble on Tuesday afternoon, feel free to ask the teacher for help (an e-mail to k.sikkel@utwente.nl will be responded to quickly).

Process

In principle you can do one iteration of the *inquiry cycle*:

- *Discover*: Have a brainstorm[5] with your project group to get a list of requirements.
- *Document*: Write these requirements down in the form of user stories, with acceptance criteria where appropriate.
- *Validate*: Interview a relevant stakeholder. Ask the interviewee for particular desires, ask what s/he thinks of the requirements that came out of your brainstorm. The interviewee can also help to establish priorities.

In reality the three phases can get somewhat mixed up. In particular when you interview someone, this will include some validation, but may very well also include new bits of discovery (which then need to be documented subsequently).

Product

The pearl assignment consists of making a requirements specification for the module project that satisfies the following conditions:

- The requirements specification is structured according to the template in Section 4.4 / Appendix C of *A primer on Requirements Analysis*.
- The requirements specification has an introduction of at least 200 words.

[5] See Skill Sheets, G3.

- The requirements specification contains at least 10 essentially different[6] requirements in the form of user stories.
- Where appropriate, acceptance criteria for the user stories have been included.
- A priority indication for the user stories has been given (for example: user stories ordered by decreasing priority / priorities indicated by means of MoSCoW categories).
- A report about validation (150–400 words) has been included as an appendix.
  Who did you interview? What did you discuss with the interviewee? Did this result in adding or deleting requirements?

As a **bonus assignment** you can extend the requirements specification as follows:

(0.3 pt) A more elaborate introduction, in which you explain to the general public what the context of the system is and which needs it is going to satisfy (at least 750 words (rather than 200)).

(0.3 pt) An appendix with a stakeholder analysis. Base this on the 'union model' (Appendix RE_A Figure 2.1). Indicate for the different roles which persons / groups / organizations will have these roles for your system. Give brief explanations, where appropriate, why you assign the stakeholders to those roles.

(0.2 pt) At least 25 (instead of 10) well-formulated essentially different user stories, with acceptance criteria where appropriate.

(0.2 pt) An appendix with screenshots (mock-up) of the system.

8.2.5 Fri 1+2, 8: Pearl assignment (closing phase)

What remains in the closing phase is wrapping up and submitting the requirements specification.

- Check wether your requirements specification satisfies all conditions stated under *Product* in Section 8.2.4.
- Most likely, different parts have been written by different persons. Read the whole document and, where needed, adapt the writing style and unify the terminology, so that it reads as one coherent document. Please check the spelling as well.
- Make a single PDF document containing the requirements specification and possible appendices. One person submits this document in Blackboard on behalf of the group.
  The deadline for submission, in principle, is Friday 23:59.
  Submission is possible until Sunday 23:59. However, requirements specifications submitted on Saturday or Sunday will not get a bonus.

[6] It is possible to get a large number of user stories by writing out all kinds of variants. For exampe: As a customer I want to buy a book / As a customer I want to buy a CD / As a customer I want to buy a DVD, etc. These are not essentially different requirements.

8.3 Example test questions

8.3.1 What to prepare

The test consists of two parts: questions about the course materials (30 %) and questions about a small case study (70 %).

- The course materials consist of
  - The two texts (*An Introduction to Project Management* and *A Primer on Requirements Analysis*).
  - The appendices *PM_A (Bennett)* and *RE_A (Alexander)*

  Read the texts one more time (probably you have studied parts of it during the execution of the project) and read the appendices. You do **not** need to learn anything by heart! You should be able to answer questions about the appendices if you have carefully read it once.
- Questions about the case study are comparable with the questions in the lecture and the lab session on Tuesday (writing a mission statement, analysis of stakeholders, formulating user stories).

Below are the original tests of 2016 and 2017. These give an idea what kind of questions will be asked—and they are useful exercise materials! The answers will be published on Blackboard one day before the test.

8.3.2 Pearl test 27 October 2016

Questions 1–3 relate to the course materials.

1 (a) What is the "iron triangle" in project management? (10 pt)
(b) How can the "iron triangle" principle help you to manage the module project next week?

2 *"The improved software for their web store should allow company RedHot to increase online sales by 15 %"* (10 pt)
Is this a goal-level requirement, a business-level requirement, a system-level requirement or a design-level requirement? Why do you think so?

3 Give five different requirements discovery techniques (10 pt)
(just names, you don't need to explain them).

Questions 4–6 relate to the case study below.

4 For a mission statement, write the paragraphs *Motivation* and *Goal of the system*. (Skip *Type of system*, *Exclusions* and *Approach*.) (20 pt)

5 The "onion-model" has stakeholder roles in multiple layers around the software system, including (but not limited to): (25 pt)
The System: Normal Operator, Maintenance Operator, Operational Support;
The Containing System: Purchaser, Functional Beneficiary, Interfacing System Owner;
The Wider Environment: Developer, Sponsor, Champion, Politician, Functional Beneficiary, Negative Stakeholder, Regulator, The Public.

   (a) Which stakeholders are mentioned in the text? To which roles do they fit?
   (b) The text did not mention any negative stakeholder. Who could be a negative stakeholder? Why do you think so?

6 Give all user stories for a customer of Ruhrstrom in Neustadt.
Give appropriate acceptance criteria for one of these user stories. (25 pt)

Case Study: Energy Demand Management

Germany has made substantial investments in renewable energy (i.e. wind and solar energy). When a substantial share of electricity production depends on these sources, this gives the complication that the production capacity varies with the weather conditions. One of the major technological challenges for the next decade is how to efficiently store and retrieve energy that is produced at peak hours.
Another way of dealing with peak levels in energy production and consumption is to induce citizens to use less electricity when supply is low / demand is high, and to shift electricity consumption to periods when supply is high / demand is low. This is called Energy demand management.
The basic idea is very simple. The price of electricity varies across the day. For each time slot of 30 minutes, the price (€/kWh) will be made available 24 hours in advance. Citizens can reduce their energy bill by shifting some of their electricity consumption to time slots with a lower price. For example for owners of an electrical car, energy demand management should be very attractive. With a simple interface you can indicate when you plan to use the car again (e.g. tomorrow 8:00), then the car will charge when electricity is cheapest.

A large pilot project for energy demand management is being set up in the town Neustadt an der Ruhr. The pilot is conducted with support of the state government of North Rhine–Westphalia as well as the Federal Ministry for Economy and Energy. Also involved is the local electricity supplier Ruhrstrom, which has to disseminate the prices per timeslot and collect all the data how much electricity was used by whom in which time slot. Neustadt is a suitable location for the pilot because all homes now have smart electricity meters that have the ability to provide Ruhrstrom with the required data. The pilot will last 12 months and will be evaluated by the state and federal governments, so as to provide input for determining future energy strategies.

The IT company BuildIT will make the new administration software that is needed for energy demand management. They will also be available for the duration of the pilot to assist Ruhrstrom and fix any bugs at short notice, might that be required. The system obviously should capture all data: electricity consumption of each household for each 30-minute time slot. Also, it should provide a web interface (for humans) and an API (for devices) giving the electricity prices for the next 24 hours. There should also be an app for smartphones.
In order to stimulate energy-aware behaviour, the city authorities insisted that Ruhstrom makes past data available to its citizens. On the website (and in the app) you can see numeric and graphic overviews of your electricity consumption the last 24 hours/week/month/3 months/year.
In order to get these data, you have to log in to Ruhrstrom's customer service site. There are two different ways to authenticate yourself when logging in: either with user ID and password, or with customer number and postal code and house number. Having logged in, you can see and download the various overviews; you can also modify your account details.

The smart electricity meters are owned by Ruhrstrom, yet they are not part of the administrative system. But they do provide input ("As a smart meter I want to transmit the electricity consumption for a given customer and time slot"). The system will also have a financial subsystem, keeping a balance for each customer, sending out debit payments (i.e. bank transfers ordered by the receiver, not the sender of the money) in regular intervals, maintaining debtor administration, etc. But to keep things short and simple we disregard that here.

8.3.3 Pearl test 26 October 2017

Questions 1–3 relate to the course materials.

1 (a) What are the four phases of a project according to PMBOK? (10 pt)
(b) If you were asked to define a fifth phases, what should it be? why?

2 In Energy Demand Management, the price of electricity varies across the day in order to discourage electriciy consumption in periods where demand is higher than supply. Possible requirements include: (10 pt)
   (a) "The system should allow the electricity company to monitor overall usage in real time;"
   (b) "The system should allow consumers to know electricity prices at least 24 hours in advance;';
   (c) "The system should allow users of electric cars to determine when it is cheapest to recharge their car."

   In each case: is it a goal-level requirement, a business-level requirement, a system-level requirement or a design-level requirement? Why do you think so?

3 (a) Give a definition of a quality requirement. (10 pt)
(b) Give an example of a constraint.

Questions 4–6 relate to the case study below.

4 For a mission statement, write the paragraphs *Motivation*, *Goal of the system* and *Exclusions*. (Skip *Type of system* and *Approach*.) (20 pt)

5 The "onion-model" has stakeholder roles in multiple layers around the software system, including (but not limited to): (20 pt)
The System: Normal Operator, Maintenance Operator, Operational Support;
The Containing System: Purchaser, Functional Beneficiary, Interfacing System Owner;
The Wider Environment: Developer, Sponsor, Champion, Politician, Functional Beneficiary, Negative Stakeholder, Regulator, The Public.

   (a) Which stakeholders are mentioned in the text? To which roles do they fit?
   (b) Give the three most essential stakeholder roles for which no stakeholder is mentioned in the text.

6 Give <u>all</u> user stories for the repair service module. (30 pt)
Give appropriate acceptance criteria for <u>one</u> of these user stories.

Case Study: Webshop Repair Service

Aaron's Audio Accessories used to be a physical shop for high-end audio equipment in Rotterdam. As the number of customers declined over the years, AAA became a webshop three years ago. Since then, business has been going up steadily. However, also increasing is the number of broken products that are returned to the service department for repair or replacement. the service department, to which customers can return products that need to be repaired, is turning into a bottleneck. In order to maintain AAA's high reputation, it is important that this service runs smoothly as well.
When equipment breaks down which has been purchased from AAA, it can be returned for repair. Most products have a warrantry period of several years. Repair within the warrantry period (or, if repair is not possible: replacement) is free of charge.
Aaron (still the CEO of AAA) sought expert advice, and a new process has been designed (see below) which should automate some clerical tasks and allow better coordination within the Service Department. In future it can be extended with an expert system to help Technical Service with the repairs, most important for now is to rationalize the process.
The next step is to construct a software module to support this process.

When a product needs repair, the customer phones the Customer Service department. Customer service staff creates a *service record* with a summary of the experienced problems. By (searching and) selecting the orginal purchase order number, essential data about the product and the customer will automatically be included in the service record. The service record will assist Customer Service and Technical Service in coordinating the repair, they can look up the service record whenever they need.
When a service record is created, the system automatically sends an e-mail to the client with the summary of the experienced problems and further instructions. Attached is a return label for the postal services, addressed to a freepost number (postage to be paid by recipient) of AAA's Technical Service department. Also, the label carries a bar code that links it to the service record.
The customer should deliver a parcel with this label to the post office.

When the parcel arrives, Technical service staff registers arrival by scanning the bar code on the label and makes sure that it gets to the right repair team.
If warrantry no longer applies, the first task is to assess what is wrong with the product, whether it can be repaired and what the estimated repair costs are. When technical Service staff adds a cost estimate (with explanation) to the service record, an e-mail with the cost estimate is automatically sent to the customer. This e-mail includes a link to a web form where the customer can indicate

- whether s/he gives permision to repair the article for the stated costs;
- if it the product is to be repaired, a bank account or credit card number to which the repair costs can be debited;
- if repair is not possible, or deemed too expensive by the customer, whether s/he would like to get the broken product returned or let AAA dispose of it in an environmentally friendly way.

When a client has given permission, technical service staff will carry out the repair. When needed, the service record can be updated with notes about the repair and costs involved. When the repair is finished, the product goes to Customer Service, who will take care of sending the product back to the customer and finalizing the service record. When Customer Service staff finalizes the service record, they may rephrase the notes of the technical service department in a more customer-friendly way. Upon finalization, an address label for a parcel is printed and the costs (up to the indicated estimate) are automatically debited to the customer's account number. If warrantry applies, the process is simpler, because no permission from the customer is needed. The article will be repaired if possible. Otherwise the 'repair' consists of replacing it with a new product.

Week 9

Project

9.1 Introduction

Theme of the week: based on the requirements specified during Pearl 111 "Requirements Engineering" (see Week 8), making a design and turning it into working software using the techniques from several earlier pearls. See Section 0.5 for a description of the desired final result.

This means that in this week you will design your system and then turn it into working software. Designing means translating the (functional) requirements that were specified in the Requirements Engineering pearl into a global design of

- *Twitter Dashboard*: ECA-rules and user-interface with the provided blocks and available data.
- *Secure Passes*: Python software on a laptop/PC and C-software on the Arduino.

A part of this is also preparing a planning: identifying the steps/actions that are necessary to deliver the artefacts. There are several deadlines you should take into account:

1. Tuesday 23:59 - delivery of design report
2. Wednesday 23:59 - delivery of preliminary system
3. Friday 13:45 - prepared presentation and demonstration (for hours 7+8).
4. Friday 23:59 - delivery of final system (including design report update)

A deadline doesn't mean that until that time you can only work on that particular task. Of course you should try to finish it as soon as possible, so you can move on to the next task. For example you could finish the design report already on Monday, so you can start building the system on Monday as well.

You will get support from teaching assistents for technical questions as well as feedback from your supervising teacher during the scheduled moments. In addition, you can of course help each other. You can post questions and problems on the discussion forum on Blackboard, but also don't hesitate to help your fellow students by answering their questions on the discussion forum.

If there is support and supervision, there is also a room. Besides that you are expected to work on the project independently as well. During those unsupervised moments you can use project rooms in various buildings (project rooms can be reserved through the WRB-system: `http://wrb.utwente.nl/`).

NB 1: We say *working* software with a reason. Starting something super ambitious and not finishing it, is not acceptable; you have to deliver something that is finished, works and has value for the envisioned

H	Mon	Tue	Wed	Thu	Fri
1	**F lec T**	F pj N	F pj N	**F pj T**	F pj N
2	**F lec T**	F pj N	F pj N	**F pj T**	F pj N
3	**F pj T**	**F pj T**	F pj N	**F pj T**	**F pj T**
4	**F pj T**	**F pj T**	F pj N	**F pj T**	**F pj T**
6	F pj N	F pj N	**F pj T**	F pj N	**F pj T**
7	F pj N	F pj N	**F pj T**	F pj N	**F tut T**
8	F pj N	F pj N	**F pj T**	F pj N	**F tut T**
9	F pj N	F pj N	**F pj T**	F pj N	

Abbr	Form
ass	Assignment
lec	Lecture
pfb	Peer feedback
prac	Practical
pj	Project
qa	Q&A
self	Self study
tst	Test
tut	Tutorial

Abbr.	Part
P	Pearl
F	Final project
S	Academic skills
M	Math

Abbr	Supervision
T	Teacher
A	Teaching assistent
N	None

Mon
- 1–2 Lecture: Project week kick-off. Preparing planning.
- 3–4 Working on the final project (with supervision).
- 6–9 Working on the final project (without supervision).

Tue
- 1–2 Working on the final project (without supervision).
- 3–4 Working on the final project (with supervision).
- 6–9 Working on the final project (without supervision).
- 23:59 **Deadline** handing in design report.

Wed
- 1–4 Working on the final project (without supervision).
- 6–9 Feedback on design reports by supervising teachers. Afterwards working on the final project (with supervision).
- 23:59 **Deadline** handing in preliminary system.

Thu
- 1–4 Feedback on preliminary system by supervising teachers. Afterwards working on the final project (with supervision).
- 6–9 Working on the final project (without supervision).

Fri
- 1–2 Working on the final project (without supervision).
- 3–6 Working on the final project (with supervision).
- 7–8 In sets of 4 groups together: presentations of final project
- 23:59 **Deadline** handing in final system + all documentation.

customer. That means that you have to give priority to realizing something that works and in minimal form satisfies the customer's requirements.

NB 2: Delivering before the deadline is obligatory. No exceptions, no delay, no excuses. This is very common for tests, presentations, project proposals, etc.; it goes for software as well. So: Not delivering before the deadline results in a 1 and therefore no final grade for the module (because the requirement is that the project grade should be at least 6). Handing in something basic which is working and presenting that on Friday is enough to stay in the race. So make sure you keep a back-up of a basic working version, so you can deliver it before the deadline if you unexpectedly can't get the actual version working. Simply a question of good risk management.

9.2 Deliverables

These deliverables are the focus this week:

Document Design report: description of design and motivation of most important design choices.
Deadline: Tuesday 23:59 for feedback by supervising teacher on Wednesday either by email or during the supervised project session.

Software Preliminary system: Source code + instructions for installation and use.
Deadline: Wednesday 23:59

Presentation Presentation/demonstration project result.
Deadline: Friday 7+8.

Software Source code + instructions for installation and use.
Deadline: Friday 23:59

Documents All project documents (new version design report; slides presentation; possible other documents)
Deadline: Friday 23:59

9.3 Week Schedule

The week schedule is based on the following phases:

- Planning and design
- First iteration, so developing a first version of the system
- Second iteration, so developing a second and also final version of the system.

Monday starts with a kick-off. In principle you could already finish the first phase with planning and design on Monday. In the week preceding the project week you have probably already done programming for the final project. You can continue that, but it is advisable to carefully think about the design. You have to deliver a report on the design which specifies the requirements and most important design choices. For the requirements you can use (modified) texts from the Trello board from the Pearl "Requirements Engineering". In addition, try to capture your design in a few insightful diagrams with corresponding explanation. Also think about what the most important design choices were and try to provide clear arguments as to why you reached that choice. Turn it into a decent professional report. It doesn't have to be very long (a few pages), as long as it satifies the goal and contains the required elements. On Wednesday your supervising teacher will give feedback on the design.

In the meantime you will continue to work on the realisation of the software. This realization happens in two phases: you deliver a preliminary version on Wednesday and a second, final version on Friday. The final version of the software should also be accompanied by a new version of the design report. You will receive feedback on the preliminary version on Thursday. Then you will also hear whether the system is already sufficient for a passing grade and, if not, what you should improve for the final system. In case

the system as delivered on Friday is still not good enough for a passing grade, you can repair it in week 10 (deadline Friday 11 November 23:59).

9.4 What you can and can't do

A situation could present itself in which you would like to change something in the original plans and requirements. A short guideline as to what you can and can't do is provided below.

CAN changing stakeholder / information request

CAN use another tweet collection

CAN "inject" tweets from other sources

CAN'T write tweets yourself (self-fulfilling prophecy)

CAN'T do something completely different than a dashboard or pass-system, but something that includes a dashboard/pass-component can be discussed.

CAN'T miss the deadline

CAN leave out or modify planned features

BAD requirements/design in design report do not match the final result

9.5 Academic Skills

There is no session for Academic Skills this week.

Week 10

Repairs

H	Mon	Tue	Wed	Thu	Fri
1					
2					
3					
4					
6			P tst T	P tst T	M tst T
7			P tst T	P tst T	M tst T
8			P tst T	P tst T	M tst T
9			P tst T	P tst T	M tst T

Abbr	Form
ass	Assignment
lec	Lecture
pfb	Peer feedback
prac	Practical
pj	Project
qa	Q&A
self	Self study
tst	Test
tut	Tutorial

Abbr.	Part
P	Pearl
F	Final project
S	Academic skills
M	Math

Abbr	Supervision
T	Teacher
A	Teaching assistent
N	None

Mon 1–8 Preparing for resit or free time.

Tue 1–8 Preparing for resit or free time.

Wed 1–4 Preparing for resit or free time.
6–9 Resits for pearls 000, 010, 100 en 110, so the tests of all uneven weeks (possibly 3 in a row).

Thu 1–4 Preparing for resit or free time.
6–9 Resits for pearls 001, 011, 101 en 111, so the tests from all even weeks (possibly 3 in a row).

Fri 1–4 Preparing for resit or free time.
6–8 Resit Mathematics
23:59 Deadline possible repairs for pearl assignments and/or final project.

Week 10 per session

10.1 Week Schedule

If you do not have any tests or assignments to repair, week 10 is completely free.

If you have to do a resit for one or more pearl tests and/or mathematics, use the time this week to prepare properly. The resits will take place on Wednesday-, Thursday- and Friday afternoon.

If you have to repair one or more pearl assignments, use the time this week to hand in those assignments. The deadline is Friday 6 November 23:59.

10.2 Academic Skills

There is no session for Academic Skills this week.