Course Outlines

Year 1

PROGRAMMING FUNDAMENTALS

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Translate an informal problem specification into a working program;

2) Solve programming challenges using the syntax and semantics of the main C/Dafny constructs;

3) Use functions and procedures to structure programs;

4) Understand the imperative implementation of some common (searching and sorting) algorithms;

5) Reason about state changes as a result of assigning values to variables (using predicates/logic) and reason about the time complexity of a program fragment;

6) Derive (small) correct programs for problems that are given by a formal specification;

7) Formally prove the correctness of given program fragments.

Description

The course 'Programming Fundamentals' is a basic introduction to imperative programming and formal reasoning about imperative programs.

The course deals with common imperative programming constructs: state (memory), assignments, conditionals, loops, functions, procedures, and recursion.

The focus in this course is on learning how to solve programming problems in a systematic way and analysis of the time complexity of the solutions. Moreover, the course focuses on formal analysis of programs, correctness proofs, and proofs of termination of loops and recursive functions.

Two programming languages are used in this course: C and Dafny.

C is used as the main implementation language for programming tasks, while Dafny is mostly used for proving the correctness of an algorithm. Testing a program can be useful to detect errors, but it is not sufficient to prove its correctness. A failed test can only show that a program is incorrect, but a passed test does not ensure that a program is correct. Therefore, we need a much more rigorous tool which is provided by the (automated) formal verifier that is supplied by Dafny.

The course should not be looked upon as a C or Dafny course. Only a subset of these programming languages is covered, that is sufficient to solve a set of small programming challenges.

Since this is the first programming course that the students take, some attention will also be paid to the Linux operating system and the available standard programs (compilers and editors). In the last week of the course, some attention is paid to the usage of library functions, and interfacing with the operating system.

Teaching method

Practical, Lecture, Tutorial

Assessment

The grade of both components (Exam, Practicals) needs to be a 5 or higher to pass the course.

If this requirement is fulfilled, then the final grade is the average of these two components.

Note that the exam itself is split up in three parts/sessions.

In case of a resit, the resit grade replaces the exam grade. The lab grade remains.

INTRODUCTION TO COMPUTING SCIENCE

General competences

Learning outcomes

At the end of the course, the student:

1) Has a global overview of the history of computer science;

2) Knows the division of computer science into fields;

3) Knows what the “grand challenges” and other key research questions in computer science.

4) Has basic knowledge of computer science concepts such as automata, languages, logic, circuits, processors, computational complexity, etc.;

5) Can implement a more complicated program than those in Imperative Programming (for example, a simulator for a simple microprocessor).

Description

This course provides a “taster” of many courses taught throughout the curriculum. Furthermore, in the second half of the course, the students will apply programming skills acquired in Imperative Programming to a larger programming problem.

As the name indicates, this course aims to give an overview of computer science, both in terms of research areas and in terms of applications. A further aim is to further mathematical skills required in computer science. The course will answer questions like:

- What is the history of computer science?

- What fields and subdivisions can be recognized within computer science (e.g., according to the ACM/IEEE)?

- What are the “grand challenges” and other key research questions in computer science?

- What are the foci of research within the Netherlands, and Groningen?

- What skills are needed to tackle key computer science problems, and how are these positioned within the curriculum?

- What are basic computer science concepts such as automata, languages, logic, circuits, processors, computational complexity, etc.?

The course also serves as an introduction to the computer science staff in Groningen, and a basic training in research skills, including both presentation skills and analytical and programming skills.

During the lectures, material from the book will be presented by different lecturers, extended by examples from research performed here in Groningen, whenever possible. During the tutorial sessions, solutions homework assignments will be presented and discussed (by students). In the first half of the course, students must read an IEEE Computer Magazine article and write an essay on it. In the second half, they must implement a larger program (in pairs).

Week 1-8 Lectures, tutorials

Week 1-4 Students write an essay on an articles from IEEE Computer as homework assignment

Week 5-8 Students work on their programming project

Teaching method

Practical, Lecture, Tutorial

We will use a mixture of lectures (2h per week), a practical in project-form (40h total), and a tutorial

Assessment

The final grade is based on written exam (40%), the programming project (20%), the essay (20%) and tutor group participation and presentation (20%). Each of the grades must be at least 5.0 to pass. If one grade is below 5.0, the maximum grade is 5.0

INTRODUCTION TO LOGIC

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Translate English sentences into Propositional Logic and First-Order Logic;

2) Analyse the validity of arguments;

3) Construct formal proofs using natural deduction for First-Order Logic;

4) Check logical/tautological consequence of sentences in Propositional Logic, using truth tables;

5) Evaluate truth of sentences in First-Order models;

6) Transform logical sentences into various normal forms (NNF, CNF, DNF, Prenex, Skolem, Horn);

7) Apply the Horn algorithm to check satisfiability of Horn sentences.

Description

This course provides a first introduction to formal logic and the development of skills in using logical symbols. The students will learn the symbolic languages of Propositional Logic and First-Order Logic, and develop skills in analyzing sentences and reasoning, assessing the validity of reasoning, making formal deductions, using formal semantics and deriving normal forms for formulas.

During the lectures, material from the textbook and the reader will be presented. During the tutorials, students can practice their skills by working on exercises, and test their understanding of the material with the help of the computer teaching program Tarski's World and with software for constructing formal Fitch-style proofs. At the tutorials, teaching assistants will be available for questions, and solutions to exercises will be discussed.

This course prepares the students for courses such as Programming Fundamentals, Discrete Structures, Languages and Machines and Computational Complexity by providing basic knowledge of logic and formal reasoning.

Teaching method

Tutorial, Lecture

Assessment

The exam grade must be at least 5.0 to pass the course.

At the first exam attempt, if Exam >= 5.0 then final grade F = max(Exam,HWME), where HWME = 8%*HW1+16%*HW2+16%Midterm+60%*Exam, ie homework's and midterm can only improve the result. At the second exam attempt, F=Exam. This is independent of whether the first exam attempt is taken at the exam or the resit.

The re-examination consists of a written exam covering the entire course material, and thereby addressing all learning goals.

COMPUTER ARCHITECTURE

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Describe the main concepts, design techniques, issues, and solutions across the abstraction layers of modern computer architectures;

2) Apply the acquired knowledge in order to solve basic design problems in computer architecture;

3) Describe the basic concepts of structured information storage and the techniques to get the information in and out of the computer.

Description

In this course we start with binary representation, transistors, logic gates & digital circuits, latches & memory circuits, and Finite State Machines. We cover the von Neumann model of computing, and then use all these tools to understand thoroughly the Instruction Set Architecture of a simple computer, the LC-3. At the same time, we will also be learning assembly language programming of the LC-3.

This course comprises two parallel tracks: i) Machine Organization, which will be developed in lecture; and ii) Assembly Language Programming, which you will learn in lab.

Machine Organization topics (Lectures):

- Microprocessors and their Instruction Set Architectures
- Overview of Assembly Language programming
- The binary representation of information: numbers, text, images, states, instructions
- Mathematical logic and Boolean algebra; transistors and simple logic gates
- Combinational Logic circuit design
- Storage circuits and memory
- Cache
- Finite State Machines
- The von Neumann model of computing
- The data paths implementing the ISA of the LC-3 processor
- Basic Input/Output (polling); memory mapping
- I/O by Interrupt
- Making the leap from assembly to Higher Level Languages

Assembly Language Programming topics (Labs):

- The LC-3 instruction Set
- Use of conditional & unconditional branches to create basic control structures
- Basic I/O
- Representing and manipulating character data
- Representing and manipulating numeric data (integer data type)

Teaching method

Tutorial, Lecture

Assessment

The final course grade is calculated by the formula $g = 0.6*e + 0.4*a$, iff $e >= 5.0$; $g=e$ otherwise, where g: final grade, e: written exam grade, a: assignments grade. A minimum of 5.0 (before rounding) on the written exam (e) is required to pass the course.

DISCRETE STRUCTURES

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Apply definitions and constructions pertaining to discrete mathematical objects such as numbers, recurrences, sets, functions, relations, partially ordered sets, graphs, and trees;

2) Count discrete mathematical objects such as sets and sequences;

3) Carry out simple mathematical proofs, including proofs by (strong) mathematical induction, to formally justify properties of discrete mathematical structures;

4) Apply various algorithms for solving problems or computing properties of relations, trees and graphs.

Description

This course covers concepts from discrete mathematics that are used to understand and describe computers and programs at an abstract level:

• Sets, sequences, languages, matrices, mathematical structures.

• Mathematical induction.

• Permutations and combinations.

• Recurrence relations.

• Partitions, relations, digraphs, equivalences.

• Functions.

• Ordering relations, partially ordered sets.

• Extreme elements, lattices, Boolean algebras.

• Trees, graphs, Eulerian and Hamiltonian paths.

During the lectures, material from the book will be presented by the lecturers. During tutorials, teaching assistants will guide students in solving exercises, and solutions will be discussed.

This course prepares the students for courses such as Programming Fundamentals, Advanced Algorithms, Languages and Machines, and Computational Complexity by providing basic knowledge of concepts in discrete mathematics and skills in mathematical reasoning.

Teaching method

Tutorial, Lecture

Assessment

The final grade is a weighted average over midterm (40%) and a final exam (60%).

The midterm will cover material from weeks 1-4. The Final exam will focus on material from weeks 5-8, but may include questions also from weeks 1-4.

The re-examination consists of a written exam covering the entire course material, and thereby addressing all learning goals. The resit grade is equal to the grade from the resit exam.

ALGORITHMS AND DATA STRUCTURES IN C

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Demonstrate knowledge of several fundamental algorithms and data structures;

2) Write efficient programs in C for the algorithms and data structures mentioned in Learning objective 1;

3) Describe algorithms in pseudocode;

4) Solve problems, using the algorithms and data structures mentioned in Learning objective 1;

5) Report in writing about assignments involving the abilities mentioned in Learning objectives 2 and 4.

Description

This course builds on the course Programming Fundamentals. In the lectures, the emphasis is on fundamental data structures (e.g. stacks, queues, linked lists, trees, graphs) and on algorithms working on these data structures. The students develop algorithms and programs related to these subjects in the practicals. Attention will be paid to 'good programming', i.e. the design of correct, efficient and understandable programs and their documentation. The programming language used in this course is C.

Teaching method

Lecture, Tutorial

Assessment

The final grade is the average of the lab grade from the programming assignments and the exam grade, provided both are at least 5.0. If one of them is below 5.0, the final grade is at most 5. Grades between 5 and 6 are rounded to the nearest integer. Higher Grades are rounded to the nearest multiple of 0.5. At least a 6 after rounding is required to pass the course.

CALCULUS 1

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Calculate limits using the limit laws, the Squeeze Theorem and l'Hospital's rule;

2) Apply the techniques of differentiation (product rule, quotient rule, and chain rule). In addition, the student can use these techniques to find the local and absolute extreme values of a given function;

3) Compute Taylor expansions of concrete functions and use these expansions to compute limits and approximations for definite integrals;

4) Apply the standard techniques of integration (substitution rule, integration by parts, trigonometric integrals and substitutions, integration of rational functions by partial fractions) and able to recognize which technique is best used in a given situation;

5) Calculate with complex numbers, can plot them in the complex plane, is able to formulate and use DeMoivre's Theorem, can compute roots of a complex number, and can solve equations in that way. The student can give the definition of the complex exponential function and can use that in applications;

6) Solve separable first-order differential equations and can compute the general solution of a first-order linear differential equation by means of an integrating factor;

7) Solve homogeneous second-order differential equations with constant coefficients and compute a particular solution for non-homogeneous equations using the method of undetermined coefficients.

Description

The course starts with refreshing and improving the basic mathematical skills (BMS). Then real functions of one real variable are treated: standard and composed functions, limits, continuity, derivatives and basics of Taylor series, extreme values, and integrals (applications and techniques). Subsequently complex numbers are introduced, an imaginary extension of the real numbers. Further, differential equations are solved to describe processes in physics, economy, medical sciences, etc. This topic includes: initial value problems, directional fields, equilibrium solutions and stability, separation of variables, variation of constants, homogeneous and inhomogeneous equations, and conversion of complex solutions into real-valued solutions. The material is illustrated through examples that focus on the applications of mathematical techniques, including basic numerical methods (such as bisection methods and Newton's method).

Teaching method

Lecture, Tutorial

Lectures, tutorials, computer-assisted tutorials to improve basic mathematical skills.

Assessment

Let G, B, WE, M and H be the final, BMS, final exam, midterm and homework average grade respectively. Then CG = (9*WE + B)/10, (on a scale from 1 to 10). If CG < 4.5, then G = CG. If CG >= 4.5, then the final grade is G = max(CG, 0.6*CG + 0.2*M + 0.2*H).

INTRODUCTION TO INFORMATION SYSTEMS

General competences

Learning outcomes

At the end of the course, the student is able to:

1) understand and know the meaning of the process of data models and the concepts of data modelling, schema modelling, the relational model, mapping of database schema to relations of a relational model, normalization, relational algebra, realization, and advantages of relational databases.

2) create a data model, define database schema and formulating queries using the SQL language.

3) Use relational database systems and SQL queries in practice.

Description

Part 1: Introduction to Databases and DB Architecture

- Databases and Database Users

- Database Systems: Concepts and Architecture

Part 2: Conceptual Data Modeling and Database Design using ER

- Data Modeling Using the Entity Relationship (ER) Model

- The Enhanced Entity Relationship (EER) Model

Part 3: The Relational Data Model, Relational Algebra and SQL

- The Relational Data Model and Relational Database Constraints

- The Relational Algebra

- Relational Database Design by ER- and EER-to-Relational Mapping

- Basic SQL

- SQL: Complex Queries, Views, and Schema Modification, Security

- Normalization

Teaching method

Tutorial, Lecture

Assessment

The final grade will be based on the written exam and the assignments.

Passing grade for written exam is 50% (before rounding).

Passing grade for assignments: none.

Passing grade for the course is 5.5 or higher.

Rounding will be done on the final grade as follows: 5.5 will be rounded to 6.0. Any grade above 6.0 will be rounded to the closest 0.5 point.

The learning objectives at the resit are tested in exactly the same way as with the original examination. The final grade is formed based on both the assignments grade (30%) and the resit examination (70%).

COMPUTER NETWORKS

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Outline and classify relevant areas and basic problems of computer networks;

2) Explain the functionality of the different layers of current network architectures. The student can reproduce, identify and select appropriate network mechanisms at each of the network layers;

3) Explain and analyze fundamental aspects of packet switching networks. The student can reproduce and identify key properties of packet switching networks and can compare their behavior with other forms of networks;

4) Acquire deep comprehension of selected, central algorithms, protocols and procedures. The student can reproduce suitable mechanisms (e.g., for media access, congestion control, addressing, or routing), explain their properties, demonstrate their behavior and compare the suitability of mechanisms in concrete network scenarios;

5) Practice writing own programs and configure networks by applying fundamental communication patterns over communication networks.

Description

Technological advancement and the trend toward increased digitization have made communication an inevitable aspect of computing systems. By now, communication systems have been established as the nerve system in most digital and physical processes in our everyday life. The goal of the course on computer networks is to provide fundamental knowledge in design and analysis of computer networks. Such knowledge is required within many application scenarios of the Internet. In this respect, the course on computer networks introduces the working principles of the Internet and its architecture by covering topics from the classic areas of computer networks, including recent developments.

At the same time, it introduces fundamental programming concepts that will enable students to write basic networked applications.

Content:

- Overview of Computer Networks (General Introduction) : Historic perspective of communications, OSI-Layers (the Open Systems Interconnection model) and Internet architecture (TCP/IP model), Performance and Quality of Service requirements (QoS) of applications over computer networks.

- Physical Layer and Link Layer: basic coding techniques, media access control.

- Network Layer: functions, service concept, an overview of basic mechanisms e.g., routing, congestion control, addressing.

- Routing protocols: foundations, classification of routing algorithms, examples of routing algorithms e.g., distance vector routing and link state routing.

- Case studies of Internet mechanisms: IP addressing and IP routing.

- Transport protocols : connectionless vs. connection-oriented transport mechanisms, e.g., handshake and flow control, case studies UDP, TCP, QUIC.

- Inter-process communications: communication taxonomy, concepts, and programming models, e.g., blocking vs non-blocking communication, concepts for SOCKET programming.

- Naming and addressing concepts in computer networks, e.g., ARP and peer-to-peer lookup mechanisms.

- Application Layer protocols, e.g., HTTP protocol

Teaching method

Lecture, Practical, Tutorial

Assessment

Students must pass the written exam with a score >=5.5; assignments and programming labs are counted if they can improve the grade of the student, i.e.,

Let E: exam grade, H: homework grade (HW assignments HW programming), G: final grade

If E>=5.5: G = max(E, 0.6*E + 0.4*H)

Else: G = E

Grades for homeworks will be acknowledged for all resits in the same study year.

LINEAR ALGEBRA

General competences

Learning outcomes

At the end of the course, the student is able to:

1) set-up, analyse and solve systems of linear equations (including examining the (in)consistency of the linear system and interpreting the solution(s));'

2) perform matrix operations (multiplication, inverse, transpose) and compute determinants;

3) formulate linear problems in the abstract setting of (normed) vector space;

4) determine properties of vectors (linear independence) and vector spaces (dimension, basis);

5) compute eigenvalues and eigenvectors and employ these in practical applications (e.g., solve systems of first-order linear differential equations with constant coefficients);

6) compute and analyse basic geometry (lines, planes, intersections, distances) and compute products of vectors (dot product and cross product) and apply them in a geometric setting

Description

This course will enable the student to:

- determine properties of vectors and vector spaces (linear independence, basis, dimension)
- use orthogonal bases and the Gram-Schmidt orthogonalisation process
- analyse and solve systems of linear equations
- set up and analyse systems of linear transformations
- perform matrix operations (multiplication, inverse)
- compute matrix determinants
- compute eigenvalues and eigenvectors and employ these in practical applications
- compute and analyse basic geometry (lines, planes, and their intersections)

Teaching method

Lecture, Tutorial

Assessment

The final grade is determined by the exam grade E, the interim exam grade IE and the average homework grade H. Specifically, the grade is computed as $G = 0.8*E + 0.1*IE + 0.1*H$. The student passes the course if G is at least 5.5. If G is less than 5.5, the student can take the resit exam and in that case their final grade is computed as $G= 0.8*RE+0.1*IE+0.1*H$ where RE is the score of the resit exam.

OBJECT-ORIENTED PROGRAMMING (FOR CS)

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Explain principles of object-oriented programming [Understanding];

2) Apply basic design patterns [Applying];

3) Identify principles of object-oriented programming applied in Java applications [Analyzing];

4) Decide when to apply OOP principles in practical applications [Evaluating];

5) Create Java applications using OOP principles [Creating].

Description

This module focuses on object-oriented programming concepts (e.g., encapsulation and inheritance) as studied in Java. We will also cover more advanced topics of Java: exception handling, IO operations, and user interface in Java.

Teaching method

Lecture, Practical, Tutorial

Assessment

Written exam: 40%, Practical: 20%, Final project: 40%. Additionally, the following constraints are applied: Each element is assessed from 0 to 10. Students need to have at least 5 for each of them. Students need to have at least 6 to pass the course.

Year 2

ADVANCED ALGORITHMS

General competences

Learning outcomes

At the end of this course, the student is able to:

1) use known strategies to design solutions to new problems, describe fundamental algorithms and data structures

2) implement a correct algorithmic solution in a programming language

3) analyse the (runtime, memory) performance of a solution and fundamental algorithms

4) evaluate a solution by means of manual and automated tests using given and hidden test inputs

5) report about activities that involve the knowledge and skills listed above.

Description

We learn a core subject in computer science related to the art of problem-solving: the practical design and theoretical analysis of algorithms and data structures.

We will look at algorithmic complexity, such as the resource consumption of an algorithm, in terms of formal analysis of computing time and memory requirements. Our approach will be analytical, focusing on asymptotic behavior: what is the increase in resource consumption when the size of the input increases? How can this asymptotic behavior be proven formally and what are typical examples of efficient algorithmic strategies to solve common problems in Computer Science?

We will deepen our understanding of such basic problem-solving strategies as Divide & Conquer, Dynamic Programming, and Greedy Algorithms. We describe a deterministic linear algorithm to find k-th-order statistics in the given array. We learn new graph algorithms. We also discuss algorithmic complexity classes and present classic NP-complete problems. We also study several ways how to cope with NP-hard problems.

These theories form the foundation of many computing systems, from web search to network protocols and efficient strategies for problem-solving. Many examples and strategies are taken from industrial coding interviews and many of the above-mentioned concepts have found their way into approximate solutions proposed in machine learning and AI.

Teaching method

Tutorial, Lecture

Assessment

The assignments (both of a theoretical nature, and some programmed in C, the A grade) are worth together 60% of the grade:

the exam (the E grade) is worth 40% of the grade. Both A and E grades must be above 5.0. You need at least 300 points in the practicals and 200 in the exam and the total sum must be >550 Points to be able to pass the course (5.5 will be rounded to 6).

The re-examination consists of an exam covering the entire course material, and thereby addressing all learning goals.

ADVANCED PROGRAMMING

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Define and identify scenarios where principles such as SOLID and Inversion of Control are beneficial to a design.

2) Illustrate a design using diagrams.

3) Classify and use software patterns effectively to address common design challenges.

4) Analyze and select appropriate lifecycles of resources such as configuration and test files in an application.

5) Integrate collaboration, building and testing tools into the software development workflow to enhance team communication and cooperation.

Description

Modern software development is more than just knowing a programming language or its libraries well. It is also about mastering the principles of software design and learning the building blocks that enable the continuous delivery of software.

This module will deal with a number of these software design principles such as software patterns and lifecycles of resources in an app. The module will also cover essential techniques and processes aimed at increasing productivity while maintaining software quality such as testing, and continuous integration.

This module is not focused on a particular programming language or paradigm and, for that reason, is not intended to teach advanced features of a particular language. That said, this module is practical in nature and will use the Java programming language and a Java technology stack to illustrate the concepts, and in the assignments.

Students are expected to have basic knowledge of imperative and OO programming paradigms. This means they are expected to have followed the Programming Fundamentals and OOP modules.

This module comprises lectures as well as practicals. Each week, one or two 2-hour lectures will explain the theory, and apply it in a 2-hour practical. This means 4 to 6 scheduled contact hours per week over a period of 6 weeks. In

Teaching method

Tutorial, Lecture, Assignment

Assessment

The course grading is a weighted sum of 2 assignments (A1: 40%, and A2: 20%) and a written exam (E: 40%). However, each component grade (A1, A2 and E) must be greater or equal to 5, or the final grade will be the smallest grade. All assignments' deliverables include code, report and demonstration of the created application.

There is a resit of the written exam. In the case of a failed assignment A1 or A2, we offer the possibility to improve the project after both assignments have been graded where old and new additional requirements must be fulfilled.

CALCULUS 2

General competences

Learning outcomes

At the end of the course, the student is able to;

1) calculate partial derivatives, directional derivatives, and the gradient, and apply this to finding maxima and minima of functions;

2) calculate contour, double, and triple integrals (in particular, find the area of a given surface);

3) solve linear second-order ordinary differential equations and solve initial-value and boundary-value problems for them;

4) decide about convergence of number sequences and series, and about the (domain of) convergence of functional sequences and series, in particular power series; expand a given function to its Taylor or Maclaurin series;

5) represent a given periodic function using its Fourier series by finding the Fourier coefficients.

Description

The course covers standard material from multivariate calculus, ordinary differential equations, and Fourier analysis. The topics to-study are: differentiation (partial derivatives) and integration (contour and surface, multiple integrals), linear ordinary differential equations (ODE) with constant coefficients and systems of first-order ODE, as well as sequences and power series (their convergence and limits), in particular Taylor and Maclaurin series and Fourier series.

Teaching method

Tutorial, Lecture

Assessment

If the exam grade is at least 4.5, then the final grade is max(exam, 0.7*exam + 0.3*homeworks), else the final grade equals the exam grade as is. Homeworks still count at the re-exam.

FUNCTIONAL PROGRAMMING

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Design algorithms in a functional style;

2) Reason formally about functional programs and proof their correctness;

3) Use polymorphism and higher-order functions;

4) Reason about the time and space complexity of programs;

5) Program in Haskell.

Description

Functional programming is a declarative programming paradigm that treats computation (execution) as the evaluation of pure mathematical functions. A function is called pure if its output is always the same for the same input and its evaluation has no side effects. As a result, expressions in functional programs can be replaced with their corresponding values without changing the behavior of the program. Functional languages are based on the use of functions that, if required, can have functions as an argument and can produce functions as a result. The typing of a function or expression is thus very important.

In functional programs, a variable within its scope has the same meaning everywhere (referential transparency). Moreover, data is immutable, which means that it cannot change after creation. A direct consequence of this property is that functional programming languages typically do not have looping constructs (for, while) since a looping variable needs to be mutable. Therefore, recursion is the only way to implement iteration.

In Haskell and some other functional languages, the value of an expression is only calculated when it is needed (lazy evaluation). This allows the construction of infinite data structures (e.g. a list of all prime numbers). Referential transparency and the lack of side effects make functional programs relatively easy to analyse mathematically. An important theme of the course is how to apply mathematical reasoning to programs, so as to prove their correctness. These proofs almost always use mathematical induction.

The aim of the module is to teach students to design and analyse functional programs in Haskell, whereby an understanding of the efficiency and general typing aspects is very important.

The course has a theoretical part which is assessed using a written exam, and a practical part that consists of assignments that need to be implemented in Haskell. The practical and the exam must be done in the same academic year.

Teaching method

Lecture, Tutorial, Practical

Assessment

Let P=practicals, E=Exam. The final grade (F) is obtained as follows: F = (3*P+4*E)/7, under the constraint that E and P both are >= 5.

In case of a resit, the resit grade replaces the exam grade. Lab grades remain the same.

STATISTICS AND PROBABILITY

General competences

Learning outcomes

1) Describe various probabilistic and statistical methods and tools;

2) Explain the difference between the frequentist and Bayesian approach;

3) Use correctly the principles of probability and statistics;

4) Analyse data and the results obtained by applying probabilistic methods and statistical tools;

5) Develop computer code to solve probability and statistics problems;

6) Assess the choice of statistical method in solving a problem.

Description

In Computer Science, Statistics and Probability are used in areas such as Algorithms, Machine Learning, and Graph Theory. Probability is the quantification of uncertainty and Statistics is the field concerned with the collection and analysis of data. This course covers the basics of Probability theory and Statistics. It discusses:

- Describing datasets numerically and visually
- Basics of Probability Theory
- Random Variables
- Discrete and Continuous Probability Distributions
- Hypothesis Testing
- Frequentist and Bayesian Inference
- Regression

A short introduction to R will be given in the first week. The lectures and the tutorials include code examples in R while the assignments can be completed in R or Python.

Teaching method

Assignment, Tutorial, Lecture

Assessment

The final score (F) is calculated as F = 0.7 * WE + 0.15 * AST + 0.15 * RS where WE is the score of the final written exam, AST is the assignment score, and RS is the Report Score. To pass the course, the exam score has to be greater than or equal to 5.5 and the average of the assignment and the report score has to be greater than or equal to 5.5.

The resit exam is similar to the original exam and the resit exam score is used as the written exam score. As for the assignment and the report, students who do not score 5.5 or more on average, will be given a single custom remake assignment from the objectives they have failed which can be used to bring the final assignment score to 5.5.

WEB ENGINEERING

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Discuss the underlying principles of the Web;

2) Show a good working knowledge of fundamental Web technologies and related concepts;

3) Design and implement a modern Web application;

4) Have the ability to locate, evaluate, and critically assess current & emerging technologies for developing Web applications.

Description

Web-based systems and applications are used to deliver a vast range of functionalities to users. Being able to properly design, develop, and maintain them is therefore very important. This course aims to address this challenge by providing students with the foundations of Web engineering.

The course covers the following subjects:

- Web and HTTP fundamentals

- Web technologies and frameworks overview

- REST fundamentals

- RESTful API design principles

- Service engineering lifecycle and principles

- Architectural considerations for Web-based systems

Teaching method

Lecture, Assignment

Assessment

The course final grade has two equally weighted components: a written exam and an assignment. The assignment can either be a project or a technology tutorial for the rest of the class. In either type of assignment, a short report is also to be delivered. A minimum of 5 (before any rounding) is required for the written exam.

SOFTWARE ENGINEERING

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Demonstrate in-depth knowledge of the initial phases of the software engineering lifecycle, i.e., requirements engineering, and apply them in a large-scale industrial setting while analysing the societal and ethical implications of the requirements;

2) Develop a software requirement specification document reflecting an in-depth comprehension of needs of an external client, considering their cultural, professional and societal background; communicate and negotiate effectively the possibilities and limitations of a product while integrating and substantiating ethical considerations in decision-making;

3) Demonstrate knowledge about collaborative software development and its implications in time management and project management (e.g., task sharing, agile practices), and reflect on the moral responsibilities of teamwork and the societal impact of collaboration outcomes;

4) Demonstrate collaborative software development and show knowledge of its implications in time management, and project management (e.g. task sharing, agile practices).);

5) Apply basic required technologies in a collaborative environment (e.g. distributed version control, continuous integration);

6) Demonstrate how to maintain a high level of quality of the software even when the project becomes large, by writing clean code and good documentation.

Description

Software engineering is the application of a systematic, disciplined and quantifiable approach to the development, management and maintenance of software. This includes activities such as project planning, project management, requirements engineering, architectural design, detailed design, implementation, validation and software evolution.

During this course students will work on a Software Engineering Project in teams of 5 members.

Teaching method

Practical, Lecture

Lectures are weekly and attendance is mandatory. Students are monitored in their attendance using a signature on the weekly attendance sheet. The level of attendance is directly used to evaluate Learninig Objective 4

Assessment

The final grade is composed of the result of the project per group, weighted by the assessment of the individual contributions. Individual contributions are evaluated using the insights gained weekly by the teaching assistants. The practical grade must be at least 5.5.

INTRODUCTION TO MACHINE LEARNING

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Understand and apply main methods and techniques used in pattern recognition and machine learning;

2) Address and solve basic problems of machine learning and pattern recognition;

3) Select appropriate techniques for particular tasks;

4) Implement basic learning and validation methods and apply them to practical data sets.

Description

The focus of this course is to introduce the concepts of machine learning, including unsupervised and supervised learning. Unsupervised learning covers dimensionality reduction, clustering, and density-based estimation methods. Supervised learning is explained mainly in terms of classification problems, with emphasis on distance-based methods such as Learning Vector Quantization as an example framework. Validation methods and the problem of over-fitting are briefly discussed in terms of classification and regression problems. Methods are also illustrated by practical applications.

Teaching method

Lecture, Practical

Assessment

Final grade: weighted average of the marks for the practicals (60%) and the written (digital) exam (40%). In order to pass, students must get a mark of at least 5.5 for both of these separately, and the weighted average has to be at least 5.75. Weighted averages above 4.75 and below 5.75 will result in a final grade 5.0, in all other cases the grade is rounded to the nearest half-integer value.

OPERATING SYSTEMS

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Outline the fundamental principles of operating systems, encompassing their evolution and milestones, features and goals, as well as architecture and constituents.

2) Describe the management of memory, input/output devices, files, processes, and threads, and assess the efficiency of different scheduling algorithms.

3) Detail the topics explored in concurrency control, such as mutual exclusion and synchronization, as well as deadlock and starvation.

4) Implement software utilizing techniques in multiprocess and multithread programming.

Description

An operating system is an essential part of any type of computer system. It functions as a mediator between the user and the computer hardware. The primary aim of an operating system is to make the computer system easy to use. A second aim is to use the hardware as efficiently as possible. The module Operating Systems introduces students to the construction of operating systems explaining various internal mechanisms with practical examples. Some of the topics that will be covered during the course include; processes and threads, scheduling, deadlocks, memory management (e.g. virtual memory, linking), storage management.

The module will be taught on the basis of the book mentioned in the course description, which will, with a few exceptions, be followed most of the time.

Teaching method

Practical, Lecture

Assessment

Let P=Practical, E=Exam. The final grade (F) is obtained as follows: if P>=5 and E>=5 then F=(P+E)/2 else F=MINIMUM(P,E)

INTRODUCTION TO COMPUTER GRAPHICS AND VISUALIZATION

General competences

Learning outcomes

At the end of this course, the student is able to:

1) Outline the mathematical, perceptual and technical foundation of digital image generation and data visualization

2) Describe and assemble a simple graphics pipeline, identify and arrange its components and explain their interactions

3) Analyse, decompose and compose visualizations in terms of techniques and design choices

4) Employ graphics algorithms, data structures and design patterns as tools to create and assess basic graphics systems and visualizations

5) Design visualizations according to visual constraints and perceptual principles

Description

This course provides students an introduction to the computational domain of Computer Graphics and Visualization. As such, the course provides the students with the knowledge to transform data into computer-generated imagery (CGI) and use the image generation to communicate insight about the data following professional- and scientific design principles. Discussing and implementing examples from virtual worlds, computer games, the natural sciences and engineering provides for an engaging context to put the knowledge into practice within the tutorials and practical assignments. By covering visual disciplines, the course also promotes creativity and develops practicable design approaches to the data presentation process, next to the technical know-how and the theoretical rationale of the domain. In preparation for advanced focus courses, the students are equipped with a toolbox of algorithms, development approaches and design tools as foundation for advanced and creative use.

A look at the human- and digital visual system, paired with a refresher of the foundational maths, are the entry points of the course. The material continues with the fundamentals of geometric modelling and object representation. Color composition, camera projection and object transformations conclude the cognitive tools for building virtual 2D- and 3D worlds. A focus on computer graphics concepts extends the fundamentals, including object illumination, image generation via Raycasting and Rasterization, and simple forms of Raytracing and Volume Raycasting, Subsequently, the courses discusses the purpose, exemplary applications and guiding use cases for visualizing data. The introduced object representations and color composition principles are employed to explain the visualization design process in an analytical, reproducible fashion. The course further illustrates how the knowledge on data, graphics and perception is used to visually encode the digital data in a communication-effective manner.

Teaching method

Tutorial, Lecture

Assessment

Exams - E (MC):

Number of exam(s): 2

Weight of Exam 1: 0.25

Weight of Exam 2: 0.75

Contribution of exams to final grade: 0.2 / 0.6

Assignments - A (AST) :

Weight per assignment to grade of A: 0.25

Number of assignments: 4

Contribution of an individual assignment to final grade F: 0.05

Total weighting of final grade - F:

F = 0.8 * E + 0.2 * A

Grading rules:

- Each individual category (E, A) >= 5.5, else F = 5.0

- F is rounded to the nearest half-integer

- If F=5.5, F is rounded to 6.0

- Resit grades replace the original grades

LANGUAGES AND MACHINES

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Understand and apply the basic theory of finite state, pushdown, and Turing machines, and of the regular, context-free, and decidable and semi-decidable languages;

2) Understand and apply the relationships between machines and languages, and the translation algorithms between the various representations (e.g. regular expressions, normal forms of grammars);

3) Apply an elementary understanding of decidability, undecidability, semi-decidability, computability, the classes P and NP, and the Chomsky hierarchy.

Description

This course addresses a fundamental topic in Computer Science: how to describe, analyze, and classify the (formal) languages that can be read by machines, as well as the machines that can read those languages.

Formal languages are specified via (1) machines for language recognition and (2) grammars for language generation. The kinds of machines employed include (non)deterministic finite state machines, (non)deterministic pushdown machines, and Turing machines.

The theme of finite state machines includes regular expressions and regular grammars, and algorithms for translating between the various forms of specification. With respect to grammars, the focus is on context-free grammars and reductions to grammars in normal form. Turing machines are used to introduce basic concepts on the decidability and semi-decidability of languages, and computability of functions. The closure properties of the classes of regular languages, of context-free languages, of decidable and semi-decidable languages are investigated. The course concludes with the Chomsky hierarchy of languages.

The course has a strong mathematical orientation in that the student must be comfortable proving or refuting plausible assertions about formal languages. The lecture notes contain many exercises, some of which will be done in the tutorials.

Teaching method

Tutorial, Lecture, Assignment

Assessment

Homework 40% and Exam 60%.

The final grade must be equal or above 5.75 for students to pass the course. The homework grade does not count at the reexamination, nor can it be transferred to future academic years.

Year 3

COMPUTER GRAPHICS

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Use basic concepts for the design, implementation, and applications of graphics systems and algorithms;

2) Describe how graphics systems work;

3) Employ the mathematical models behind graphics system operations.

Description

This course introduces the main concepts behind two-dimensional and three-dimensional computer graphics such as scan-conversion, transformations including the necessary linear algebra, the rendering pipeline with its stages, local and global illumination models, and shape representations. Lectures are accompanied by tutorials and practicals on OpenGL and ray-tracing, and the relevant theory and algorithms. These tutorials and practicals deepen the understanding of the presented material through discussions and implementation. The course is closely related to image processing, computer vision, scientific visualisation, linear algebra, and object-oriented techniques, and is a prerequisite for the Intelligent Systems and Visual Computing MSc specialisation at the RUG.

Teaching method

Lecture, Assignment, Tutorial

Attendance at the first practical session is mandatory.

Assessment

If P1>=4 and P2>=4, then P=0.5*P1+0.5*P2, else P=MIN(4,0.5*P1+0.5*P2), where P1/P2 is the average of two OpenGL/Raytracer assignment grades. The final grade F: If P>=5 and E>=5, then F=0.5*P+0.5*E, else F=MIN(5,0.5*P+0.5*E), where E is the exam grade. The subgrades are not rounded, but the final grade is rounded to the nearest half-integer with the exception of 5.5, which is rounded to 6.

RESEARCH SKILLS IN COMPUTING SCIENCE

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Employ basic bibliography search techniques in order to systematically identify literature on a given subject;

2) Review and summarize literature in order to provide the state of the art in a given area;

3) Analyze the state of the art on a subject in order to formulate appropriate research questions that require further investigation;

4) Synthesize all the above elements to produce a complete research proposal, including methods and planning.

Description

This course aims primarily to train Computing Science students in developing the research skills necessary for the completion of their Bachelor's Project (WBCS901-15). These skills are also essential for students that are interested to further work in contexts where writing, analyzing, and defending a research proposal are required activities, such as working in a research-intensive organization (either in the industry or academia).

Students taking the course are asked to:

1) Select one of the topics made available to them by the CS staff members, or propose one of their own;

2) Use search techniques (search engines and digital libraries, snowballing, etc.) to expand the bibliography provided by the supervisor of the selected topic;

3) Critically read the resulting bibliography and summarize the current state of the art in their topic direction;

4) Formulate a research question that could form the basis of their Bachelor's project, together with a plan to address it;

5) Based on the above steps, write a full research proposal for their BSc thesis project.

Topic supervisors are responsible for instructing and coaching through this process. There is at least one round of peer and supervisor reviewing of the proposal for comments and corrections. Students are encouraged to continue with the same topic for their Bachelor's project, but can switch to another one, always in consultation with the respective topic supervisor.

Teaching method

Assignment, Lecture, Tutorial

Active participation in the discussion sessions (which take place during lecture slots) is obligatory. Deadlines of assignments are hard and not negotiable.

Assessment

Each student starts with a 10 for P and loses 2 points for not attending a mandatory session, 1 point per incomplete review, and 1 point per day late per submission. S is the proposal grade given by the student's topic supervisor. C is based on reviewing two MSc Student Colloq. presentations. The final grade is 0.3*P + 0.6*S + 0.1*C; both P and S have to be at least a 6 to pass the course.

OPERATING SYSTEMS

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Outline the fundamental principles of operating systems, encompassing their evolution and milestones, features and goals, as well as architecture and constituents.

2) Describe the management of memory, input/output devices, files, processes, and threads, and assess the efficiency of different scheduling algorithms.

3) Detail the topics explored in concurrency control, such as mutual exclusion and synchronization, as well as deadlock and starvation.

4) Implement software utilizing techniques in multiprocess and multithread programming.

Description

An operating system is an essential part of any type of computer system. It functions as a mediator between the user and the computer hardware. The primary aim of an operating system is to make the computer system easy to use. A second aim is to use the hardware as efficiently as possible. The module Operating Systems introduces students to the construction of operating systems explaining various internal mechanisms with practical examples. Some of the topics that will be covered during the course include; processes and threads, scheduling, deadlocks, memory management (e.g. virtual memory, linking), storage management.

The module will be taught on the basis of the book mentioned in the course description, which will, with a few exceptions, be followed most of the time.

Teaching method

Practical, Lecture

Assessment

Let P=Practical, E=Exam. The final grade (F) is obtained as follows: if P>=5 and E>=5 then F=(P+E)/2 else F=MINIMUM(P,E)

BACHELOR'S PROJECT

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Define, prioritize, restructure, clarify, refine, and categorize general problems and requirements of the supervisor;

2) Apply his acquired computer-science knowledge, making use of scientifically and technically founded methods, tools, and processes;

3) Cope efficiently and effectively with change during the project's lifetime, considering the goals and requirements of the supervisor, and also the consequences of the student's decisions;

4) Critically read, study, and decide the applicability of subject-matter literature and other technical information in the context of his/her assignment, and also considering other constraints (e.g. deadlines and complexity);

5) Critically assess the results of his/her own work and the work/results of other parties;

6) Work in a project fashion, in a team and/or in close contact with the supervisor(s);

7) Present the achieved results (technique, research, design, implementation, project steps, project results) adequately in writing and orally, both to a technical and non-technical audience;

8) Demonstrate professional behavior: reliability, involvement, independence, precision, and ability to follow and respect (intermediate) deadlines.

Description

A Bachelor’s thesis project has the following general characteristics:

- A project has a project owner (typically BI staff member, in special cases external person);

- A project has two supervisors (of which the so-called main supervisor is a BI staff member);

- A project covers 15 EC points (420 hours of study, one full block);

- A project is finalized by a BSc thesis and an oral presentation;

- Project work is divided into several steps (different for different concrete projects). These are discussed with the project's supervisor(s);

- The typical length of a BSc thesis is 30-40 pages (of course, the exact length depends on many aspects, including typesetting and use of imagery);

- The oral presentation (typically, PowerPoint/slides supported) takes typically 20 minutes. The audience include other BSc students, and the BI staff (supervisor(s) but also other staff members);

- The final evaluation and grade is done by the project supervisor(s).

Teaching method

Practical

The project work, the BSc thesis and the oral presentation are mandatory

Assessment

Pass mark in consultation with the supervisor. The project work, the BSc thesis and the oral presentation are mandatory.

COMPILER CONSTRUCTION

General competences

Learning outcomes

At the end of the course, the student is able to:

1. Design a (small) imperative programming language and implement a compiler for it;

2. Use code generation tools (scanner generators and parser generators);

3. Apply theory (from a course like Languages and Machines (WBCS027-05)) in a practical setting.

Description

For most users, a compiler is a black box that translates a source program into machine code. In this course, we open this black box and discuss all phases of compilers (and their implementation) for imperative programming languages.

These main phases are:

1. Lexical analysis phase

2. Syntax analysis phase

3. Semantic analysis phase

4. Intermediate code generation phase

5. Code optimization phase

6. Final code generation phase

Teaching method

Practical, Lecture

Assessment

All practicals exercises must have been completed with a grade > 5. Moreover, the exam and the practical part must have a grade >= 5.5. Let P be the mean grade of the practicals, and E the exam grade. The final grade F is computed as F=(E+P)/2.

In case of a resit, the Exam grade is replaced by the Resit grade. Lab grades remain.

COMPUTATIONAL COMPLEXITY

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Explain the design choices that underpin classical complexity theory (the focus on time/space resources, asymptotic worst case complexity, polynomial/exponential classes);

2) Apply the basic concepts in computation and complexity such as Turing machines, (un)decidability, time and space complexity classes;

3) Prove properties concerning the various complexity classes and prove the simpler main theorems;

4) Construct reductions from one problem to another for suitable examples in order to classify decision problems into complexity classes.

Description

Computational complexity aims to classify computational problems based on the amount of resources (in this course: time and space) that is required by the best algorithm to solve the problem. This topic may be viewed as the mathematical study of efficient computation. It is distinct from the analysis of specific algorithms although the topics are related.

The standard choice of computational model in this field is the Turing machine. After a brief review of Turing machines, decidability and undecidability, we examine some of the main ideas and results including the time complexity classes P and NP, the famous unsolved (open) question of P versus NP, polynomial time reducibility, NP-completeness of SAT and other NP-complete problems, space complexity classes like NL and PSPACE, Savitch's theorem, NL=coNL, oracles, and the hierarchy theorems which serve to separate certain complexity classes e.g. P is a proper subset of EXPTIME.

This is a course in the field of Theoretical Computer Science.

Some of these topics appear in Languages and Machines (WBCS027-05) but those topics are explored here at much greater technical depth.

Teaching method

Tutorial, Lecture

Assessment

The assessment consists of two written assignments. Each assignment is completed individually by the student. The student must submit the two assignments on time, obtain at least 3 in each assignment, and achieve a weighted grade of at least 5.5, calculated as the average of the two assignment grades.

A student qualifies for an extra (Resit) assignment if the weighted grade of the two assignments is less than 5.5, and at least 3 was obtained in each assignment. In the event of a Resit, the end grade is minimum (6.5, average of the two highest assignment grades)

INFORMATION RETRIEVAL

General competences

Learning outcomes

After completing the course Information Retrieval, students will be able to:

1) Understand the role and the importance of information retrieval;

2) Understand the basic principles, and methods behind information retrieval;

3) Design and develop an information retrieval system, and report its specifications;

4) Understand how to read, interpret, and summarize results of IR systems;

5) Present the main contributions and results of state-of-the-art literature in IR.

Description

With the ever growing availability of data in different forms, the ability to retrieve information of interest is becoming more and more important. Information retrieval (IR) is the process of finding the right data from an information system that is relevant to a given query. It is, thus, the science of searching for information in textual, visual and audio data, as well as metadata of such files. Automated IR systems are needed to deal with the phenomenon of information overload. One of the most popular examples of IR systems is a web search engine.

The first step of an IR system is a user query. A query is a formal statement of an information need, e.g., search string in web search engines. It may not uniquely identify a specific object (an entity that is represented by some information) in a collection. Various objects may satisfy the query to different extent/relevance. In contrast to classical SQL queries of an RDBMS, IR results may not precisely match the query, and therefore they are ranked according to some matching criteria. This is a key difference of IR systems in comparison to database searching.

The course consists of a theoretical and a practical part. During the theoretical part, the main aspects of Information Retrieval will be discussed, including an overview of the field as well as the core algorithms. The practical part consists of the design, development of IR systems, as well as execution and reporting of assignments in groups. This activity will be supported by computer labs and mentoring by teaching assistants.

Topics that will be covered include:

- Introduction: The IR problem, system and the web

- Boolean retrieval and term vocabulary

- Scoring, term weighting and the vector space model

- Indexing and searching

- Retrieval evaluation
- Relevance feedback and query expansion
- Web retrieval and crawling
- Multimedia information retrieval

Teaching method

Tutorial, Lecture

Assessment

The assessment consists of an exam (50%) and a set of assignments (50%). The grade of the latter is an aggregate score of the group assignments (including presentation). Equal contribution of members within the same group will be assumed for each assignment, unless otherwise indicated by the students.

INFORMATION SECURITY

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Select and apply various encryption/decryption and hashing methods;

2) Use the PGP/GPG encryption software;

3) Explain the strengths and weaknesses of various protocols;

4) Select best authorization model/technology to a given setting;

5) Contrast different anonymization and privacy enhancing technologies;

6) Discuss various security testing methods such as fuzzing.

Description

Information security refers to protection of sensitive data against unauthorised access and misuse. This introductory course covers both theoretical and practical aspects of Information Security, trying to lay the foundations of security infrastructures we use on a daily-basis. Following a general introduction to the thoughts behind Information Security (e.g., CIA), various aspects of cryptography, access control, security protocols and privacy protection will be discussed. In the context of cryptography, stream and block cyphers will be contrasted and widely used encryption methods (e.g., RSA, AES) as well as hashing algorithms (e.g., SHA- 1/2, Tiger) will be presented in detail. Some of these topics will be supported with accompanying homework assignments.

In addition to cryptography topics, the concepts of authentication and authorisation along with different access control models such as MAC and RBAC, and the supporting technologies such as OAuth2 and OIDC will be presented. Since most of the concepts covered in the course are part of security protocols, common security protocols that are widely used such as SSL will be studied. The last two lectures will be on recent but established topics on privacy protection (e.g. k-anonymity, differential privacy) and system security (e.g. binary analysis, fuzzing) going beyond traditional Information Security topics.

The course will also have lab sessions which will focus on practicals about, among others, encryption/decryption, Diffie-Hellman key exchange methods and fuzzing. The labs will mainly serve as a supporting tool for the assignments.

Teaching method

Lecture, Assignment, Practical

Assessment

The written exam (E), combined with the ratings of assignments (A) produce the final grade: if (A >= 5.0 and E >= 5.0) then F= (.6*A + .4*E) else otherwise (A<5 or E<5) F=min(A,E))

NOTE: We employ Themis in the submission/evaluation of assignments so if you are not knowledgeable about Themis contact us at the very beginning of the course.

INTRODUCTION TO IMAGE PROCESSING

General competences

Learning outcomes

At the end of this course, the student is able to:

1) Identify, evaluate and use images with different channels, quantizations, sampling resolutions and frequency spectra

2) Measure entire images as well as their content for their enhancement, analysis, feature summary and comparison

3) Create image filters by means of statistics, frequency, morphology and derivatives to enhance image appearance and detect meaningful features

4) Decompose image content into foreground/background and individual discrete- space binary large objects via segmentation

5) Reduce images with respect to user-constraints and target capacity via compression

Description

This course introduces students to essential theoretical concepts for processing digital images, and equips them with practical experience in mapping the concepts on a multitude of application cases. A central aspect of image processing is the design, implementation, improvement and application of filters to enhance images (by removing noise and sharpening outline edges), transform images to alternative spaces (e.g. Fourier space), and extract content-specific feature information, such as regions and edges. Another major topic is the formulation of interpolative geometric transformation to scale, translate and rotate images, as well as to project them to different view planes. Segmentation of image content into fore- and background, as well as the separation and labeling of different image objects, is another core domain in image processing. Lastly, principles of information theory, frequency transformation and perception are used to compress images in lossy- or lossless manners effectively, in order to reduce their memory footprint and prepare them for network-channeled transmission. Those concepts, and a selection of cutting-edge image processing topics, are taught in this course via lectures and practical assignments.

Teaching method

Lecture, Practical

Attendance of the 1st lecture is mandatory.

Assessment

Exam - E:

Minimal passing grade: 5.5

Number of exam(s): 1

Weight of the exam grade E: 0.7

Assignments - A:

Minimal passing grade per assignment: 5.5

Weight per assignment to grade of A: 0.33

Number of (graded) assignments: 3

Contribution of an individual assignment to final grade F: 0.1

Total weighting of final grade - F:

F = 0.7 * E + 0.3 * A

Normal rules:

- Each individual category (E, A) >= 5.5, else F = 5.0

- F is rounded to the nearest half-integer

- If F=5.5, F is rounded to 6.0

- Resit grades replace the original grades

KNOWLEDGE TECHNOLOGY PRACTICAL

General competences

Learning outcomes

To acquire and develop skills in the field of knowledge systems. To design and construct a knowledge system on the basis of expert knowledge.

Description

In the course, students design and construct a knowledge system based on expert knowledge. The course addresses rule-based representation and inference. Expert interviews, project planning, system evaluation, and reporting and presentation of the knowledge system are a part of the course. Attention is paid to techniques for knowledge acquisition and designing a user interface.

Teaching method

Tutorial, Lecture

The project presentation meeting is mandatory.

Assessment

The grade elements (report and assignment) are combined in one standard overall grade on a scale from 1 to 10. The submitted code and presentation are used in the grading of the project report.

LANGUAGE TECHNOLOGY PRACTICAL

General competences

Learning outcomes

At the end of the course, the student is able to:

understand transformer architectures' core concepts and components, including self-attention mechanisms, multi-head attention, and feed-forward networks.

Develop proficiency in implementing and fine-tuning transformer models for specific NLP tasks.

Develop practical coding skills in working with transformer-based libraries and frameworks such as Hugging Face's Transformers, PyTorch, or TensorFlow.

Apply transfer learning principles to leverage pre-trained transformer models for downstream NLP tasks, improving efficiency and performance.

Description

The Language Technology Practical course aims to provide students with hands-on experience working with transformer-based models for Natural language processing tasks. Transformers have had a significant impact on natural language processing (NLP) and have played a crucial role in achieving state-of-the-art results in various tasks, including machine translation, language understanding, text summarization, and generation. This course will primarily focus on understanding and applying transformer models in different language technology applications. students will learn effective ways to utilize pre-trained transformer models and adapt them to specific NLP tasks using transfer learning techniques. By the end of the course, students will have gained the practical knowledge and skills necessary to work with transformer-based models and apply them to real-world natural language processing problems.

Teaching method

Practical, Lecture

Assessment

To pass the course, students must complete all assignments. Additionally, to receive a passing grade, the final project must have a score higher than 5.

PROCESS-AWARE INFORMATION SYSTEMS

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Understand and discuss the main concepts of process management, BPM life cycle, architecture of Process-aware Information Systems (PAIS), and principles of building process-based applications;

2) Apply the acquired knowledge in order to model and develop process-based solutions for application building and/or application integration;

3) Describe the basic concepts of BPM and be able to compare them to other application programming or integration approaches, as well as identify and assess the advantages and disadvantages of BPM in concrete application areas.

Description

The course starts with an introduction to the foundations of Business Process Management (BPM) and Process Re-engineering. The BPM methodology and life cycle, the Basic principles of building Process Aware Information Systems (PAIS) are presented and examples are given of a large variety of application areas. The historical development of PAIS is considered too.

Based on the introduction as a framework for the whole course, the definition of all the life cycle phases is given thus explicitly identifying all activities necessary to build and execute a process-based application. The phases include process modelling, IT refinements, implementation and deployment of processes, process execution, monitoring and process mining. Subsequently the life cycle is used to identify the components of a PAIS and their role towards enacting computerizes processes. Special attention is paid on the process modelling phase and the available principles. The de facto standard for process modelling BPMN 2.0 will be presented.

The course continues with presentation and discussion of the process execution phase from the perspective of the workflow itself as well as from the perspective of the enactment system, so-called workflow management system or workflow engine. Here we will learn about how process instances are executed and how external software and/or people are used as implementations of tasks in a process. Finally a service-based process execution language - BPEL - is presented and will be used for the implementation of service-based processes, also called service orchestrations.

Teaching method

Practical, Lecture

Assessment

The final grade is the exam grade. Passing grade for the course: 5.5 and higher.

PROGRAMMING IN C++

General competences

Learning outcomes

After successfully completing this part of the C++ course you are able to:

1) Create production-grade C++ programs. The student has a mental model of what goes on 'under the hood', and can argue the code's clarity, maintainability and efficiency;

2) Construct classes that handle dynamically acquired resources properly. In addition to using STL containers, they will overload operators when necessary, and write exception-safe functions;

3) Create single-purpose classes to compose their programs of, judging whether to use composition, inheritance, polymorphism, friends, entirely separate classes or any combination thereof;

4) Reduce time spent coding by making use of STL facilities such as generic algorithms, and improve efficiency by deploying multiple threads of execution;

5) Extend the clean coding style of C++ Fundamentals to larger and more robust programs.

Description

This external course is organized and taught by CIT. This is an intensive course, aimed at students who already possess some C++-specific skills.

Programming in C++ is the middle of three courses that together cover the language. It builds upon the proper handling of acquired resources taught in C++ Fundamentals with STL containers, more operator overloading and exception handling. It also treats composition, inheritance and polymorphism in class design, and some parallel computing (through multi-threading and forking). The primary focus is on the language and the Standard Template Library.

The teachers take the stance that besides knowledge, a programmer needs skills and good habits. The course builds those by requiring assignments to be completed every week. Within two weeks, the student has to go from understanding the topics to skillfully applying them.

Considerable attention is given to writing in a maintainable style: e.g. sensible and consistent layout, comments, const-correctness, single-purpose functions, and classes.

Having completed this course, students are able to write production-grade programs in C++, although advanced techniques like Templates and Concepts are only taught in Advanced C++.

Teaching method

Lecture, Assignment, Tutorial

Lectures: 2hr, homework exercises: approx. 15 hr.

Assessment

A 56% average score on the assignments is required to sit for the exam. To pass the course, the student must pass the exam. A good exam can increase the final score by up to a full point. The student can skip the OE if they follow up with Advanced Programming in C++. Then a single OE after that will result in a grade for the two (or even three) courses combined.

Pass is a grade 6 for the exam and a 56% or better score on the assignments. The assignments offer practice in writing code. The exam also tests understanding, both in general and applied to specific cases. E.g.: What's the purpose of making virtual member functions private? Would the shown class benefit from being made move-aware?

The assignments are group work, mostly programming exercises and some insight questions. Both correct answers and style are important. The OE serves to verify that the student has not (even unintentionally) been leaning too much on other group members.

The group work is graded. If enough time is left before the deadline, one resubmission is allowed, which usually greatly enhances the score.

REINFORCEMENT LEARNING PRACTICAL

General competences

Learning outcomes

At the end of the course, the student is able to:

1. Explain how the Reinforcement Learning framework applies to complex decision-making problems.

2. Identify common issues that arise from environment interactions and implement strategies to address them.

3. Create a proposal for a Reinforcement Learning project.

4. Develop a Reinforcement Learning project under guidance.

5. Reflect critically on the design, implementation, and outcomes of a completed Reinforcement Learning project.

Description

This course covers the practical aspects of implementing reinforcement learning (RL) methods to solve complex decision-making problems.

Students will work on a project, beginning with ideation and problem framing, followed by environment design or selection, and progressing through policy design/selection, training and hyperparameter tuning, model evaluation, and presenting a trained policy in a demo environment.

While there will be lectures to introduce projects and discuss the RL framework and tools used to develop them, most of the course will focus on hands-on practical work with RL environments and algorithms.

Guidance on potential project ideas will be provided; however, students are encouraged to bring their own ideas that align with their interests, passions, or specific areas of knowledge, allowing them to explore RL in meaningful, personalized contexts.

The Reinforcement Learning Practical course is not an introductory course on reinforcement learning; therefore, we will not have lectures on RL theory. Instead, the emphasis will be on RL libraries and tools needed for RL projects (such as Gymnasium, PyTorch, and TensorBoard).

Teaching method

Lecture, Practical

Mandatory attendance:

- (Virtual) attendance during group meetings with Teaching Assistants is required. Part of the final grade is determined by demonstrating the progress on the project, which is assessed in these meetings.

- Final presentation

Assessment

The final grade is determined by a weighted average of:

- 20% assignment on multi-armed bandits

- 10% project proposal

- 10% progress meetings

- 15% final project oral presentation

- 35% final project report

- 10% reflection assignment

At least a 5.0 for both the final project report and the reflection assignment are required. The weighted average of all assignments needs to be at least a 5.5.

SHORT PROGRAMMING PROJECT

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Implement a larger programming or research project;

2) Turn an informal specification of a more complex problem into an exact list of requirements;

3) Analyse these requirements and turn them into an effective design or research experiment;

4) Implement the code and properly document it;

5) Phase the project;

6) Report on the project in written and oral form.

Description

This course is entirely in project form and therefore requires a lot of initiative and independence from the participating students. The aim is to learn how to tackle larger, real-world programming problems individually. During this project the student must develop a software product either for one of the research groups in the university or for a company. A limited number of projects will be made available to be taken at a first-comes-first-serves basis, leading to a limit in enrolments for this course. However, students may also propose their own projects, found in collaboration with a company for example, which is then subject to approval by a staff member allowing them to be taken from the waiting list and being enrolled into the course. In the former case, the project will be supervised by the member of staff who has proposed the project, in the latter the student is responsible for finding a supervisor. In all cases the specifications of the project are highly individual and to be agreed upon before the start of the project by everybody involved.

After selecting or proposing a project, the student must draw up a project plan, in which the informal specification must be translated into a precise formulation of the requirements. The plan and specification of requirements must be approved by the supervisor(s). During the teaching period (half semester) the student must work on analysis, implementation, and documentation of the software product. The student must demonstrate the product to the supervisor at the end of the period, and a report must be handed in. Often the supervisor will ask for a presentation of the final results as well.

The supervisor determines the grade based on requirements elicitation, analysis, code efficiency, correctness, maintainability, and documentation, the level of independence, handling of project planning, the final report and the presentation.

Teaching method

Practical

Assessment

The student has to pass from both the report and the implementation, where s/he needs to obtain at least 6 in each of them.

SIGNALS AND SYSTEMS (FOR CS)

General competences

Learning outcomes

At the end of the course, the student is able to:

1) Mathematically analyse continuous-time signals;

2) Mathematically analyse and practically implement linear filters;

3) Implement and use (fast) Fourier transforms for 1D and 2D signal processing;

4) Do frequency-domain analysis on signals via Fourier transforms.

Description

This course is an introduction to (mainly digital) signal processing and discrete linear systems theory. The module consists of lectures and tutorials/practicals.

The topics that are covered in the course include:

- Continuous-time and discrete-time signals
- Periodic signals: Fourier series
- Sampling
- Linear time-invariant (LTI) systems/filters
- Convolution, convolution theorem and Fourier transform
- Time domain and frequency domain analysis
- The Z transform
- FFT algorithms (and their derivations)

During the tutorials, the theoretical concepts will be made more concrete by making exercises (math!) on paper.

During the labs the theory is applied by making small programs. These programs will be written in C, so the student is expected to master this programming language.

Teaching method

Practical, Tutorial, Lecture

Assessment

To be entitled to enroll for the exam, each lab session must have been made and the grade for each lab session is at least 5.5. Let P=mean lab grades, and E=grade written exam. The final grade F is obtained via the rule: if E<5.5 then F=E else F=(E+P)/2.

ADVANCED LOGIC

General competences

Learning outcomes

At the end of the course, the student is able to:

1. Define sets inductively and to prove properties of natural numbers and well-formed formulas by induction.
2. Check and explain whether given inferences are valid in fuzzy logic.
3. Check whether given inferences are valid in several three-and four-valued logics, using truth tables and tableaux.
4. Check whether given inferences are valid in several modal logics, including temporal logics, using tableaux.
5. Check whether given inferences are valid in default logic, using process trees.
6. Present soundness and completeness proofs of tableau systems for several modal logics.

Description

In this course, students learn that there are several other logics than the classical propositional and predicate logic that they know from the introductory logic course. Indeed, each context and application area requires a developer to think about which type of logic is appropriate to model the given situation, for example, in order to prove a program to be correct or to design communication protocols among software agents.

The topics of the course include: proof by induction on the natural numbers and on the construction of well-formed formulas; many-valued logics, including three-valued, four-valued, and fuzzy logic; several types of modal logic, including propositional and first-order modal logic, as well as temporal logics; the method of tableau proofs for propositional logic and several modal logics; soundness and completeness proofs; and nonmonotonic reasoning, especially default logic, including its procedural semantics.

Teaching method

Lecture, Tutorial, Practical

Practical work consists of independently making homework exercises and peer reviewing others' homework exercises.

Assessment

Written, Exam

Assignment

ADVANCED PROGRAMMING IN C++

General competences

Learning outcomes

After successfully completing this part of the C++ course you are able to:

1) Design and use function templates as well as class templates;

2) Explain the template parameter deduction process;

3) Explain why Substitution Failure Is Not An Error (SFINAE); and to explain how this principle affects template selections;

4) Define (partial) specializations for class templates; and to explain why function templates do not support partial specializations;

5) Design and implement variadic templates;

6) Use tuples to return structured types from functions, and use them to define differently typed variables in the initialization sections of for-, if-, and switch statements;

7) Define statically polymorphic classes, preventing the software from having to use dynamic or late binding;

8) Use template meta programming to convert values to types, to define the organization of classes based on its template parameters, and to use compile-time selection and iteration without creating any run-time executed code;

9) Use template concepts to specify the requirements of template type arguments;

10) Use precompiled headers and/or modules to reduce compilation times and to avoid the use of include guards in header files;

11) Use transactional memory to avoid having to protect statements executed in multiple threads by mutexes;

12) Use code generators instead of designing lexical scanners and syntax parsers ‘by hand’.

Description

This external course is organized and taught by CIT. Continuing after Programming in C++ participants are at the level where they are familiar with all standard and predefined elements of the C++ programming language and are able to design understandable and maintainable large-scale C++ programs. Advanced Programming in C++ continues from there, and strongly focuses on compiler performed code generation.

Teaching method

Lecture, Assignment, Practical

Assessment

Assignment

Practical

Presentation

COGNITIVE MODELLING PRACTICAL

General competences

Learning outcomes

At the end of the course, the student is able to:

1. Program a cognitive model

2. Validate a cognitive model on the basis of empirical data

3. Use cognitive models to make predictions for new experiments

4. Describe the findings of cognitive models in a written report

Description

This course provides hands-on experience with computational cognitive modelling using the cognitive architecture ACT-R (http://act-r.psy.cmu.edu/). Previous basic knowledge about ACT-R will be necessary (BSc course Architectures of Intelligence/ACT-R tutorial). In this course we will be developing computational cognitive models to improve our understanding of cognitive processes. We will cover the entire process: formulating a hypothesis, implementing it in a cognitive model, evaluating the model's predictions, and finally communicating your findings through a presentation and a scientific report.

Teaching method

Practical, Lecture, Tutorial

Assessment

Students must pass the report to pass the whole course (pass grade for the report is 5.5). The overall pass grade is 5.5

AGENT TECHNOLOGY PRACTICAL

General competences

Learning outcomes

At the end of the course, the student is able to:

1. Design an agent-based model to answer a research question about real-life multi-agent systems with simple agents, such as traffic systems, animal group behaviour, and contagion spreading.

2. Argue for and against design choices made in the construction of an agent-based model.

3. Implement an agent-based model, with simple agents, in NetLogo.

4. Modify existing agent-based models in NetLogo to better answer existing research questions, and to answer novel ones.

5. Run an appropriate simulation experiment using agent-based models in NetLogo in order to answer a research question.

6. Accurately and convincingly report the results of a simulation experiment using agent-based models.

7. Critically appraise the validity of conclusions drawn from their own and other's agent-based models in NetLogo.

Description

This course provides an applied introduction to the field of agent-based modeling (ABM). By analysing, extending, and designing agent-based models, you will obtain a deep understanding of the practical uses of ABM and the skills to construct an agent-based model from scratch. In the final group project, you will make use of ABM to investigate a subject of choice.

The lectures present advantages and disadvantages of ABM techniques, practical implementations of these techniques, and critical reflection on their outcomes. The structure, progression, and outcomes of these ABMs are presented using NetLogo.

Teaching method

Tutorial, Lecture

Assessment

The final grade is based on three homework assignments and a group project. The group project consists of implementing an agent-based model, writing a project report, and giving a presentation.

The final grade is calculated as 0.1*ASM1 + 0.1*ASM2 + 0.1*ASM3 + 0.7*PR, where ASMn is the n-th homework assignment, and PR is the grade on the final project.

As additional requirements, the final project grade must be at least a 5.5 to pass this course, and the final project presentation must be assessed as sufficient in order to obtain a course grade. In case of a failing project grade, resits consist of an opportunity to improve the final project.