Course Outlines
Year 1
FOUNDATIONS OF PROGRAMMING FOR COMPUTER SCIENCE
Module aims
Students will be introduced to different programming constructs, basic data structures, command line tools, integrated development environments and unit testing of programs. Students will learn how to describe well-defined tasks using pseudocode and translate them into programs using a procedural programming paradigm. The module will be taught using a procedural language for practising these skills.
Module learning outcomes
S101 Describe and apply the fundamental concepts of procedural programming. Write small procedural programs from scratch to perform well-defined tasks, following well-defined requirements, in a procedural programming language like Python. Relate the syntax of the language to its semantics, and analyse the result of executing fragments of syntax. Integrate library code with their own programs using appropriate software tools.
S102 Implement bespoke data structures to store states of a process. Implement simple algorithms written in pseudocode. Develop programs incrementally, using simple tests (automated where appropriate) to check each increment.
S103 Store data in memory in standard built-in collection types, and to store and retrieve data from simple text files such as CSV and JSON files.
S104
Use an appropriate software development environment, such as Eclipse, IntelliJ or VS Code. Given a program and a debugging tool, students will be able to identify and correct bugs which prevent the program from functioning as intended.
S105
Organise and document program code following the principles of software engineering. Write documentation to explain the design and implementation of their own code, or example code which is supplied to them.

Indicative assessment
Task % of module mark
Online Exam -less than 24hrs (Centrally scheduled) 100
Special assessment rules
None

Indicative reassessment
Task % of module mark
Online Exam -less than 24hrs (Centrally scheduled) 100
Module feedback
Feedback is provided through work in practical sessions, formative assessments, and after the final assessment as per normal University guidelines.

Indicative reading

Allen B. Downey - Think Python: How to Think Like a Computer Scientist - 2nd ed. (2015), O'Reilly Media

Mike Dawson - Python programming for the absolute beginner - 3rd ed. (2010) - Course Technology

Kent D. Lee and Steve Hubbard - Data structures and algorithms with Python (2015), Springer

MATHEMATICAL FOUNDATIONS OF COMPUTER SCIENCE

Module aims

Students will be introduced to the key discrete mathematics concepts that are the foundation of computer science. Seven topics are covered as follows: i) counting (combinatorics), ii) discrete probability, iii) graphs, iv) propositional and predicate logic, v) proofs and sets, vi) relations on sets, and vii) relations on a single set. After studying the module, students will be able to apply the learnt concepts, theories and formulae in real-world examples of computational problems.

Module learning outcomes

T101 Define, read and apply mathematical notations for the purpose of describing mathematical concepts from across discrete mathematics.

T102 Select appropriate techniques to prove properties related to discrete mathematics concepts.

T103 Understand how to construct sets of elements with certain properties and determine their cardinality using counting formulae from combinatorics.

T104 Understand and apply basic set theory, including formally defining set relations and operations.

T105 Describe and use the basic concepts of discrete probability to describe events, with an understanding of joint, conditional and marginal probabilities, Bayes' theorem, expectation, covariance and correlation.

T106 Formally define and illustrate by example graphs of different graph classes, such as simple, undirected, directed, weighted, directed acyclic, connected, disconnected and trees - with an understanding of how they may be used in real-world computational problems.

T107 Apply a variety of techniques to identify whether logical expressions are true or false, valid or invalid or equivalent to one another, and be able to apply logical statements to describe real-world logical problems.

Indicative assessment

Task	% of module mark
Closed/in-person Exam (Centrally scheduled)	100

Special assessment rules

None

Indicative reassessment

Task	% of module mark
Closed/in-person Exam (Centrally scheduled)	100

Module feedback

Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines.

Indicative reading

** Dean N., The Essence of Discrete Mathematics, Prentice Hall, 1997

** Haggarty R., Discrete Mathematics for Computing, Addison Wesley, 2002

** Truss J., Discrete Mathematics for Computer Scientists, Addison Wesley, 1999

** Gordon H., Discrete Probability, Springer, 1997

* Solow D., How to Read and Do Proofs, Wiley, 2005

HUMAN-COMPUTER INTERACTION

Module summary

HCIN introduces user-centred design. Where other modules focus on technical understanding of computers and how they work, this module is instead about understanding the relationship between computer systems and people. It discusses how this can be used to improve system development, and how it can go wrong. We will explore the nature of and barriers to people's interactions with computers and how systems can be designed to optimise and facilitate these interactions. We will also consider how to evaluate the people's experience - what makes a good, enjoyable human-computer interaction.

Related modules

The assessment is undertaken as a group open assessment in which groups will work together to complete and present a user-centred design task. The assessment also includes as part of the submission an individual reflection exercise which is worth 10% of the overall marks. This task asks students to comment on their development of transferable skills (using the York Strengths Framework as a basis) and how they have developed these during HCIN. The assessment also has a peer-assessment exercise, which ensures that individual contributions to the group are accounted for in the final marks.

Reassessment is by an individual open assessment which requires students to understand and implement the processes and techniques discussed during the module.

Module will run

Occurrence Teaching period

Module aims

Students taking this module will be familiarised with how to design user-centred systems that meet the needs and preferences of diverse users. Students will be introduced to the notion of engineering lifecycles, and in particular building requirements from user needs, iterative prototyping and evaluation of interactive systems. Students will undertake group work in practicals, giving them opportunities to develop communication and conflict resolution skills. The assessment will evaluate knowledge of the user-centred design process and interaction design principles.

Module learning outcomes

Describe why user-centred design in software development is important to usable and inclusive design.

Undertake a user-centred design process as a cyclical approach through the key stages of user needs elicitation, conceptual design, prototyping, and evaluation.

Apply appropriate interaction design concepts in describing user-system interaction including: affordances, feedforward, feedback, conceptual model.

Advocate for the ethical treatment of participants throughout the user-centred design lifecycle, and explain how user diversity can impact on the inclusiveness of a system.

Describe how interactive systems are embedded in societal structures, and how they are used to invoke change at the personal, community, national or international level.

Plan and manage deliverables to set deadlines throughout a project lifecycle, and use a self-reflective skills assessment to improve student team working and team performance.

Indicative assessment
Task % of module mark
Essay/coursework 10
Groupwork 90
Special assessment rules
None

Indicative reassessment
Task % of module mark
Essay/coursework 100
Module feedback
Feedback is provided through work in practical sessions, formative assessments, and after the final assessment as per normal University guidelines.

Indicative reading
*** Preece, J., Rogers, Y., Sharp, H., Interaction Design, 4th edn Wiley, 2015

*** Cooper, A., Reimann., R., Cronin., D., Noessel., C. About Face: The Essentials of Interaction Design. 4th edn Wiley, 2014.

* Mackenzie, I.S. Human-Computer Interaction. Elsevier Inc., 2013.

* Norman, D. The Design of Everyday Things. Any edition.

OBJECT-ORIENTED DATA STRUCTURES AND ALGORITHMS

Module aims

Students begin to program key data structures such as stacks, queues, trees and graphs. They are introduced to the idea of complexity of an algorithm, and how to characterise time and space through formal notations and proof techniques. Students are taught using an object oriented language like Java, and learn the basics of test driven development for testing their code and demonstrating its successful running. Students are also introduced to several algorithm design paradigms such as greedy algorithms.

Module learning outcomes

S201

Implement an object oriented design. This includes organising program code into modules using methods following the software engineering principles of modularity and abstraction, and assembling data and methods into classes at an introductory level following the software engineering principles of encapsulation and data hiding. Integrate standard library code with their own programs using appropriate software tools.

S202

Write and test code that conforms to specific interfaces. Identify and correct bugs which prevent the program from functioning as intended.

S203

Organise and document program code following the principles of software engineering. Generate documentation, manually and programmatically, to explain the design and implementation of their own code, or example code which is supplied to them.

S204

Analyse problems in order to confidently design algorithms to solve simple problems,and be able to explain how algorithms and Processing programs work. Develop small programs that implement basic algorithmic designs. Argue the correctness of algorithms using inductive proofs and invariants.

S205

Analyse worst-case running times of algorithms using asymptotic analysis and apply the knowledge to sorting and searching algorithms, categorising efficiency in time and memory use.

S206

Compare between different abstract data structures from linked lists to graphs in order to be able to choose an appropriate data structure for a design situation. This includes the major search, sort, and graph algorithms and their analyses.

S207

.Apply algorithmic design paradigms such as greedy and dynamic programming paradigms. Present an argumentation for the choice of a paradigm for a given problem. Describe what an approximation algorithm is, the benefit of using approximation algorithms, and analyse the approximation factor for such an algorithm.

Indicative assessment

Task	% of module mark
Online Exam -less than 24hrs (Centrally scheduled)	50
Online Exam -less than 24hrs (Centrally scheduled)	50

Special assessment rules
None

Indicative reassessment

Task	% of module mark
Online Exam -less than 24hrs (Centrally scheduled)	50
Online Exam -less than 24hrs (Centrally scheduled)	50

Module feedback
Feedback is provided through work in practical sessions, formative assessments, and after the final assessment as per normal University guidelines.

Indicative reading
Steven S. Skiena - The algorithm design manual - 2nd ed., (2010), Springer

Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser - Data Structures and Algorithms in Java (2014), Wiley Etextbooks

Mitsunori Ogihara - Fundamentals of Java programming (2018), Springer

INTRODUCTION TO COMPUTER ARCHITECTURES
Module will run
Occurrence Teaching period
Module aims
Students taking this module will gain foundations in the key architectural components of a computer system, how a high level program is executed upon that system, and how a computer system is constructed in hardware. Students will be introduced to how a computer system is constructed, and how a program implemented in a high level programming language (for example C) executes on that system. Students will be introduced to the basics of computer architecture and program language construction, providing a basis for further study in later years. Students will be introduced to a bottom-up approach, motivated by real examples, taught as both lectures and laboratory practicals. Students will be able to describe and apply their programming skills on real devices and computer systems that are used in many real applications today.

Module learning outcomes
Identify the purpose of key computer hardware components such as processors, memories and busses.
Describe different data types commonly found in binary systems (e.g. signed vs. unsigned integers), and show how to convert, perform arithmetic, and perform logical operations.
Express logical expressions as basic gates, transistors and combinatoric logic circuits
Describe the function and limitations of a variety of logical building blocks in the context of processor architectures
Describe the von Neumann Model paradigm of computer architecture, including the fetch execute cycle of instruction processing.
Explain how operations executed in a processor can be used to implement to higher level sequential, conditional and iterative programming language constructs
Build a simple system comprised of a CPU, memory and input/output.
Explain the use of assemblers, compilers and linkers to create executable code for a processor, and use such a toolchain develop software for the simple system built in the module
Identify potential security problems associated with architecture design.
Indicative assessment
Task % of module mark
Essay/coursework 100
Special assessment rules
None

Additional assessment information
Open assessment is a long-running task that takes place over a number of weeks.

Indicative reassessment
Task % of module mark

Essay/coursework 100
Module feedback
Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines.

Indicative reading
*** J.Hennessy, D.Patterson Computer Architecture: A Quantitative Approach (2nd Edition) Morgan Kaufmannn 1990

*** W.Stallings Computer Organization and Architecture: Design For Performance (8th Edition) Pearson 2010

FORMAL LANGUAGES AND AUTOMATA
Module will run
Occurrence Teaching period
Module aims
Students taking this module will be introduced to the concepts of formal languages and the abstract machines that accept them as a way of describing computation. Students will have a deep understanding of finite automata and pushdown automata, with their associated languages and related proof techniques, and will be introduced to more complex machines accepting context sensitive and recursively enumerable languages for purposes of being able to identify and describe them.

Module learning outcomes
Describe and illustrate the concepts of formal languages, automata and grammars, and the relations between them;
Construct a variety of abstract machines including: deterministic and non-deterministic finite automata, deterministic and non-deterministic pushdown automata and Turing machines;
Distinguish different classes of automata, and the languages they accept;
Apply a variety of operations to transform and convert between automata;
Convert between grammars and automata for regular and context-free languages;
Demonstrate that a grammar is ambiguous;
Apply the pumping lemma for regular and context-free languages to show a language is not regular or context-free respectively;
Describe the Chomsky hierarchy;
Identify key applications in computing where regular and context-free languages are used in practice; and
Use automata theory as the basis for building lexers and parsers.
Indicative assessment
Task % of module mark
Open Examination 100
Special assessment rules
None

Indicative reassessment
Task % of module mark
Open Examination 100
Module feedback
Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines.

Indicative reading
*** Peter Linz, An introduction to formal languages and automata. Sixth Edition. Jones and Bartlett Computer Science. 2017
** D.I. Cohen, Introduction to Computer Theory. 2nd Edition. Wiley. 1997

** Hopcroft, John E. and Motwani, Rajeev and Ullman, Jeffrey D., Introduction to Automata Theory, Languages and Computation (3rd ed.), Pearson Education, 2013
** S.H. Rodger and T.W. Finley, JFLAP: An interactive formal language and automata package. Jones and Bartlett Computer Science. 2006. Available online
* Martin, John C., Introduction to Languages and the Theory of Computation (4th ed.), McGraw Hill, 2010
* Rich, Elaine, Automata, Computability and Complexity, Pearson Education, 2008
* Sipser, Michael, Introduction to the Theory of Computation (3rd ed.), South-Western College Publishing, 2012

Year 2
DATA: INTRODUCTION TO DATA SCIENCE

Module aims
Students will be introduced to key concepts required to undertake rigorous and valid data analysis. Students will be introduced to processes for collecting, manipulating and cleaning data, while gaining experience in judging the quality of data sources. Students will be introduced to statistical analysis in data science, including correlation, inferential statistics and regression, and how to use these tests in a programming environment. Relational databases, SQL, and and other database paradigms such as NoSQL, are covered as a way of storing and accessing data. A key aim of the module is to solve complex problems and deliver insights about multi-dimensional data.

Module learning outcomes
Distinguish between different types of data that are generated in science, engineering and design, and employ strategies for ensuring data quality.
Retrieve data from a variety of different data sources in a variety of different formats.
Apply inferential statistics and statistical procedures to test hypotheses about features and relationships within data sets.
Use appropriate visualisations to present and explore data sets.
Use databases, both relational and of other paradigms, to store and query data.
Identify the ethical concerns regarding the provenance of data, the privacy of individuals, and the impact data analytics can have on society, and apply topics from the code of ethics of a professional data protection body.

Indicative assessment
Task % of module mark
Essay/coursework 100
Special assessment rules
None

Indicative reassessment
Task % of module mark
Essay/coursework 100
Module feedback
Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines.

Indicative reading
*** Spiegelhalter, D., The Art of Statistics: Learning from Data, Pelican, 2019.

*** VanderPlas, J. Python Data Science Handbook: Essential Tools for Working with Data, O'Reilly, 2016.

** Igual, L. Segui, S. Introduction to Data Science: A Python Approach to Concepts, Techniques and Applications, Springer, 2017

SYSTEMS & DEVICES 2: OPERATING SYSTEMS, SECURITY, AND NETWORKING

Related modules

Pre-requisite modules

Systems & Devices 1: Introduction to Computer Architectures (COM00011C)

Co-requisite modules

None

Prohibited combinations

None

Module aims

This module builds on Systems and Devices 1 by examining the system software that executes upon a computer system. Students will learn how the resources of the system can be shared by multiple programmes and users, and how networking can be used to communicate between programmes. One important aspect is how basic security and protection mechanisms are provided by the processor and memory system. Throughout, the module students will consider practical examples based on computer systems used today. This module also introduces students to the core concepts of computer networking by covering the layered network model, and discussing the utility and motivation for such an approach. Services that are layered on this model (such as UNIX sockets, DNS, TCP, IP) are detailed and students will develop software to experiment with these features. After taking this module, students will have an understanding of the role of an operating system, how computers can support multiple time-sliced programmes, and how all kinds of computer networks, including the Internet, are created.

Module learning outcomes

Demonstrate application programming of OS-supported concurrency, communication and I/O. Show how the structure of the OS is supported by computer hardware, with specific reference to the hardware features that extend the basic systems introduced in S&D1.

Use basic resource management mechanisms provided by common OSes, including time and memory.

Demonstrate use of the memory protection mechanisms provided by hardware and OSes, including memory mapped I/O.

Demonstrate use of the information security provided by the OS in terms of file systems.

Demonstrate concurrent programming at the process level and show how it is supported by, and implemented on, the system hardware.

Learn to recognise and avoid issues of deadlock, livelock, and starvation.

Be able to articulate the motivation behind the layered network model

Develop software using OS-level networking concepts (i.e. sockets) to communicate with other systems.

Demonstrate understanding of networked architectures, how they are integrated into an operating system, and develop simple applications using this knowledge.

Indicative assessment
Task % of module mark
Essay/coursework 50
Online Exam -less than 24hrs (Centrally scheduled) 50
Special assessment rules
None

Additional assessment information
Students are only required to resit any failed assessment component.

Indicative reassessment
Task % of module mark
Essay/coursework 50
Online Exam -less than 24hrs (Centrally scheduled) 50
Module feedback
Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines.

Indicative reading
*** Operating System Concepts, Tenth Edition by Silberschatz, Galvin and Gagne, Wiley (2018)

* A. S. Tanenbaum, Modern Operating Systems, Prentice Hall (2014).
* W. Stallings, Operating Systems, Internals and Design Principles (9th Edition) Ninth Edition, Prentice Hall (2017).

THEORY 3: COMPUTABILITY, COMPLEXITY & LOGIC

Module aims

This module covers computability theory and complexity theory. In particular, students will learn the concepts of semi-decidable and decidable languages, and Turing-computable functions. They will be able to explain the difference between solvable and unsolvable problems and prove unsolvability by reduction. They will understand the time and space complexity of Turing machines, complexity classes such as P, NP, PSpace, NPSpace and NPC, and prove NP-completeness by reduction. The module will also introduce basic concepts and results in propositional logic, predicate logic, and program verification. In particular, students will learn to distinguish between syntax and semantics, and be able to use formal proof systems such as natural deduction. They will understand the limitations of logic in terms of decidability and expressiveness, and how to use a formal calculus such as Hoare logic to specify programs and prove them correct.

Module learning outcomes

Use unrestricted grammars and Turing machines to specify semi-decidable languages.

Provide examples of unsolvable problems and prove that a problem is unsolvable by reducing a known unsolvable problem to it.

Explain the Church-Turing thesis and its significance.

Define the classes P and NP, and explain their relation to the class ExpTime.

Explain the significance of NP-completeness and provide examples of NP-complete problems.

Explain the meaning of formulas in propositional and predicate logic, and translate such formulas into English and vice-versa.

Explain the fundamental difference between syntax and semantics.

Apply the rules of natural deduction to construct proofs, and determine the truth or falsity of formulas in a given model.

Explain the limitations of logic and the relationship between logic and computability.

Reason deductively about programs using formalisms such as Hoare logic and weakest preconditions

Indicative assessment

Task	% of module mark
Closed/in-person Exam (Centrally scheduled)	100

Special assessment rules
None

Indicative reassessment
Task % of module mark
Closed/in-person Exam (Centrally scheduled) 100
Module feedback
Feedback is provided through work in practical sessions, revision classes, and after the final assessment as per normal University guidelines.

Indicative reading
**** Martin, John C., Introduction to Languages and the Theory of Computation (4th ed.), McGraw Hill, 2010

** Rich, Elaine, Automata, Computability and Complexity, Pearson Education, 2008

** Sipser, Michael, Introduction to the Theory of Computation (3rd ed.), South-Western College Publishing, 2012

* Hopcroft, John E. and Motwani, Rajeev and Ullman, Jeffrey D., Introduction to Automata Theory, Languages and Computation (3rd ed.), Pearson Education, 2013

* Arora, Sanjeev and Barak, Boaz, Computational Complexity: A Modern Approach, Cambridge University Press, 2009

++ Garey, Michael R. and Johnson, David S., Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, 1979

INTELLIGENT SYSTEMS: MACHINE LEARNING & OPTIMISATION

Module aims

This module introduces the field of Artificial Intelligence, key approaches within the field and philosophical questions such as what it means for a machine to understand. Students will learn the theory and practice of machine learning techniques covering linear regression, simple neural networks, linear algebra and continuous optimisation. Students will see motivating real world problems, the ML techniques required to solve them, the underlying mathematics needed for the technique and their practical implementation. Practicals will be taught using Python, and the group project will introduce the students to a Python-based modern machine learning library such as TensorFlow or PyTorch.

Module learning outcomes

Explain the difference between strong, weak and general AI, understand the relationship between computation and AI, define the machine learning paradigm, and distinguish it from the wider field of AI

Compute partial derivatives and understand the concept of the gradient as a generalisation of the derivative

Express, manipulate and solve systems of linear equations using linear algebra, and apply linear regression and logistic regression

Optimise multivariate functions using gradient descent

Explain the concept of overfitting and how regularisation can be used to prevent it

Construct a basic neural network using a modern machine learning library and learn its weights via optimisation using the backpropagation algorithm

Deconstruct ethical arguments relating to AI and its applications, and appreciate the ethical and privacy implications of machine learning

Indicative assessment

Task	% of module mark
Essay/coursework	30
Online Exam -less than 24hrs (Centrally scheduled)	70

Special assessment rules
None

Indicative reassessment

Task	% of module mark
Essay/coursework	30

Online Exam -less than 24hrs (Centrally scheduled)70
Module feedback
Feedback is provided through work in practical sessions, and after the assessments as per normal University guidelines.

Indicative reading
Artificial Intelligence: A Modern Approach by Russell and Norvig

ENGINEERING 1: SOFTWARE & SYSTEMS ENGINEERING
Related modules
Pre-requisite modules
Software 2: Object Oriented Data Structures & Algorithms (COM00016C)

Co-requisite modules
None

Prohibited combinations
None
Module aims
This is the students' first opportunity to integrate their skills into a development project. Starting from a broad problem description and working in groups, students will design, develop and test a complex system. The students will be introduced to the software engineering terminology, lifecycle and processes and will become familiar with principles, techniques and tools for, and develop hands-on experience of eliciting requirements; defining software architectures; designing and implementing software in an object-oriented way using established patterns; reviewing, testing and refactoring software systems; and setting up continuous integration and delivery processes. Students will also develop an appreciation of how to identify, mitigate and monitor risks, how to manage software projects, and how to reuse and extend 3rd-party code/libraries. Overarching themes of the module will include traceability, cyber-security and ethical considerations across the engineering lifecycle.

Module learning outcomes
E101

Apply an understanding of software engineering terminology, lifecycles and process models, to help with undertaking a project.

E102

Elicit and document user and system requirements.

E103

Architect, design and implement software in an object-oriented way.

E104

Demonstrate how the estimation of risk can be used to improve decision-making, and to make realistic estimates for a project.

E105

Define unit- and system-level tests for software, and use continuous integration processes.

E106

Apply mechanisms for working in teams to successfully undertake a group project.

E107

Apply different models for software licensing and reuse of 3rd party software to the artefacts developed in the module.

E108

Demonstrate consideration of cyber-security and ethical considerations in the engineering lifecycle, through practice and documentation.

E109

Write documentation and design/deliver presentations for the different stages of the engineering lifecycle.

Indicative assessment

Task	% of module mark
Groupwork	35
Groupwork	65

Special assessment rules
None

Indicative reassessment

Task	% of module mark
Essay/coursework	35
Essay/coursework	65

Module feedback
Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines

Indicative reading
**** Ian Sommerville, Software Engineering (latest edition), Addison-Wesley, 2010

*** Roger Pressman, Software Engineering (latest edition), McGraw-Hill, 2010

SYSTEMS & DEVICES 3: ADVANCED COMPUTER SYSTEMS

Module aims

This module continues the Systems and Devices stream by considering advanced computer systems in terms of their structure and how it affects their programming. To this point, the systems introduced have been basic uniprocessors with simple memory. The module starts by introducing the concept of multiple processor architectures, pipelined and superscalar processors, systems-on-chip and advanced memory structures, including caches.

The rest of the module considers how these hardware features affect the way that such modern systems are efficiently programmed. Issues of programming for pipelines, caches, etc. are considered. Then how processes running in parallel on separate processors can share underlying resources safely. Finally, the module considers building blocks for parallel programming, e.g. threads and communication. It will also consider how such constructs map to processor instructions. The module will consider real-world examples throughout.

Module learning outcomes

S301

Appreciate how, and why, the hardware structure of multiprocessor architectures differs from the architectures discussed in S&D1. Use this knowledge to develop software applications for such architectures.

S302

Use knowledge of computer system design concepts to evaluate the likely performance of a proposed computer architecture.

S303

Demonstrate the ability to optimise software for advanced hardware features like caches, branch predictors, and pipelines.

S304 Demonstrate through software development, how data structures are laid out in memory, how they can be accessed from a program, and how processor instructions are generated to manipulate them.

Indicative assessment

Task	% of module mark
Online Exam -less than 24hrs (Centrally scheduled)	100

Special assessment rules

None

Additional assessment information
Students will only need to retake failed assessment components.

Indicative reassessment
Task % of module mark
Online Exam -less than 24hrs (Centrally scheduled) 100
Module feedback
Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines

Indicative reading
**** H. Abelson, and G.J. Sussman, Structure and Interpretation of Computer Programs, MIT Press, 1996

*** R.W. Sebesta, Concepts of Programming Languages, Addison Wesley, 2009

*** A. Burns and G. Davis, Concurrent Programming, Addison-Wesley, 1993

** D. Watt and W. Findlay, Programming Language Design Concepts, Wiley, 2004

** T.W. Pratt and M.V. Zelkowitz, Programming Languages: Design and Implementation, Prentice Hall, 2001

** D. Lea, Concurrent Programming in Java, Addison Wesley, 1996

** A. Burns and A. Wellings, Concurrent and Real-Time Programming in Ada, CUP, 2007

** F. Casarini and S. Thompson, Erlang Programming, O'Reilly, 2009

** A. Burns and A. Wellings, Real-Time Systems and Languages 4/e, Addison Wesley

Year 3
CAPSTONE PROJECT (PRBX): COMPUTER SCIENCE (UG)
Professional requirements
Please see the additional assessment information above.

Module will run
Occurrence Teaching period
Module aims
The aims of this module are to provide a culmination of three years' teaching in a substantial Capstone Project Module (CPM).The project provides an introduction to independent study in an engineering context, supports synthesis and application of material from the degree course, and gives the student the opportunity to demonstrate an appreciation of engineering methods and techniques, through coverage of requirements, ethical considerations, specification, design, implementation and evaluation. On completion of the project, the student will have gained the practical skills that can only be gleaned from the experience of undertaking independent (supervised) study. The student will also have the experience of having written a substantial academic report.

Module learning outcomes

Demonstrate acquired specialisation in a particular part of the subject area, including enhanced or new technical skills that build on taught theory.

Demonstrate acquired skills to undertake a computer systems (software and/or hardware) engineering project, including design, implementation and evaluation.

Demonstrate a practical understanding of how established techniques of research and enquiry are used to create and interpret knowledge.

Recognise alternatives, selecting and justifying the approach taken at each point in the report, identifying parts of the project area that are feasible within the time (etc) constraints of the project.

Appreciate the latent issues of the subject area (for example, in software engineering they might meet and tackle such as emergent requirements, design flaws, equipment/application problems).

Prepare a written report on the work done, according to the defined criteria. In particular, the student should be able to prepare a report with a good structure and clear presentation, and in which the referencing is of publishable academic standard. The report must demonstrate critical abilities and evaluation of work done and methods applied.

Articulates an understanding of legal, ethical, social, professional and commercial issues involved in the project, detailing potential issues and mitigation strategies.

Summarise the context, method, results, and implications of the project in an engaging form for a non-expert audience.

Indicative assessment
Task % of module mark
Essay/coursework 100
Special assessment rules
Non-compensatable

Additional assessment information
Please note, as per University of York assessment regulations (see the Rules for Progression and Award), it is only possible to resit UG Capstone Project Modules (CPMs) when this is permitted within the total module credit resit threshold for the year. Currently, this threshold is 60 module credits.

Due to PSRB (accreditation requirements) there are also special assessment rules that apply to PRBX. First PRBX cannot be compensated. Second, only in the case of marginal fail of PRBX (marks that fall within 30-39), reassessment is permitted. Students are given the opportunity to make amendments to enable them to reach a pass threshold, within a specified time frame. The mark for the resubmitted PRBX will be capped at the pass mark (40). There will only be one such reassessment. These rules are applicable to all students who take PRBX, regardless of programme.

Indicative reassessment
Task % of module mark
Essay/coursework 100
Module feedback
Feedback on written report draft (where draft provided to supervisor in a timely manner). Written feedback after written project report.

Indicative reading
*** Dawson, C. W Projects in Computing and Information Systems. Addison-Wesley 2005

*** Gowers, E. The complete plain words. Penguin 1987

*** Kopka, H and Daly, P.W. A guide to LATEX : document preparation for beginners and advanced users, 3rd edn. Addison-Wesley 1999

*** Zobel, J. Writing for computer science, 2nd edn. Springer 2004

AI PROBLEM SOLVING WITH SEARCH AND LOGIC

Elective Pre-Requisites

These pre-requisites only apply to students taking this module as an elective.

A Level Maths or Equivalent, plus good knowledge of AI fundamentals, predicate logic and programming.

Module will run

Occurrence Teaching period

Module aims

This module will introduce key approaches in Artificial Intelligence for tasks such as: finding a sequence of actions to achieve a goal; playing adversarial games; and solving discrete optimization problems such as configuration and scheduling. Students will learn the theory and practice of AI search, logic, and constraint-based approaches. The module aims to equip students with a wide range of problem-solving tools, how to design effective heuristics for them, and enable comparison of methods to determine which are best suited to a given problem. Some of the tools covered are state-space search algorithms (i.e. A* Search, IDA*, and Greedy Best-First Search), game-tree search algorithms (i.e. Minimax and Monte-Carlo Tree Search), local search methods for solving discrete optimization problems, constraint programming, and the satisfiability (SAT) problem in knowledge representation and reasoning.

Module learning outcomes

Represent a given search problem in terms of states, actions, and a goal, and identify a suitable heuristic.

Represent a given scenario using propositional logic to enable logical inference (for example, using a SAT solver).

Model (represent) and solve discrete optimization problems using a modern constraint programming system.

Select and apply an appropriate AI state-space search algorithm for a given problem, identifying reasons for the choice of algorithm in comparison to others.

Select and apply an appropriate adversarial (game-tree) search method to solve a given game, including design of a suitable heuristic if required.

Describe the algorithms commonly used in SAT (propositional satisfiability) solvers, local search solvers, and constraint solvers, and apply them to small examples.

Indicative assessment

Task % of module mark

Closed/in-person Exam (Centrally scheduled) 100

Special assessment rules

None

Indicative reassessment

Task	% of module mark
Closed/in-person Exam (Centrally scheduled)	100

Module feedback

Feedback is provided throughout the sessions, and after the assessment as per normal University guidelines.

AUTONOMOUS ROBOTIC SYSTEMS ENGINEERING

Module aims

This module will introduce students to the theoretical concepts and practical skills required to engineer autonomous robotic systems. It will cover fundamental aspects of sensors/actuators and control systems, then build upon this foundation with high-level algorithms for autonomous localisation, mapping, navigation, and multi-robot coordination. This module will also explore safety considerations and ethical implications of the design, implementation, and deployment of autonomous robotic systems.

Module learning outcomes

Describe the degrees of autonomy that robotic systems can achieve

Discuss the safety considerations and ethical implications of the design, implementation, and deployment of autonomous robotic systems

Explain methodological principles for engineering autonomous robotic systems

Demonstrate an understanding of modern robotics middleware and its application

Implement an autonomous robotic solution for a predefined problem

Discuss the strengths and weaknesses of an implemented autonomous robotic solution

Indicative assessment

Task	% of module mark
Essay/coursework	100

Special assessment rules

None

Indicative reassessment

Task	% of module mark
Essay/coursework	100

Module feedback

Feedback is provided throughout the sessions, and after the assessment as per normal University guidelines.

Indicative reading

Correll, Nikolaus, et al. Introduction to Autonomous Robots: Mechanisms, Sensors, Actuators, and Algorithms. MIT Press, 2022.

Herath, Damith, and David St-Onge. Foundations of Robotics: A Multidisciplinary Approach with Python and ROS. Springer Nature Singapore, 2022.

EVOLUTIONARY & ADAPTIVE COMPUTING

Module summary

This module introduces a range of biologically-inspired approaches to computing.

Module aims

This module introduces a range of biologically-inspired approaches to computing. It provides a foundation of both theoretical and practical knowledge on the subject of evolutionary computation, an optimisation technique inspired by biological evolution. Students will have hands-on experience implementing a number of types of evolutionary algorithms using Python and the library DEAP: Distributed Evolutionary Algorithms in Python, to solve a range of different types of problems. The module also studies the use of Agents and Multi-agent Systems as a modelling paradigm, with a focus on evolutionary adaptation and learning.

Module learning outcomes

Design and implement evolutionary systems to address a given problem.

Understand the biological underpinnings of evolutionary algorithms, and use them to optimise mathematical functions and agent behaviours.

Define a range of agent behaviours and represent them in a form that is well suited to natural selection

Model processes in populations of agents using hand-written mathematical models

Critically evaluate the performance and implementation of evolutionary and multi agent systems.

Indicative assessment

Task	% of module mark
Essay/coursework	100

Special assessment rules

None

Indicative reassessment

Task	% of module mark
Essay/coursework	100

Module feedback

Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines.

Indicative reading

++ Banzhaf et al, Genetic Programming: An Introduction, Morgan Kaufmann , 1999

++ M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998

COMPUTER VISION & GRAPHICS

Module summary

This module will introduce modern computer vision approaches, including discussion of the main applications and challenges.

Module aims

This module will introduce modern computer vision approaches, including discussion of the main applications and challenges. It will cover issues of image formation, camera geometry, feature detection, motion estimation and tracking, image classification and scene understanding, using a range of model-based approaches.

Module learning outcomes

Demonstrate a detailed understanding of the image formation process, its modelling in computer vision and its simulation in computer graphics

Describe and implement techniques for rendering images including modelling light/material interaction

Understand a range of methods for inferring 3D shape from images

Apply fundamental machine learning methods for image understanding

Indicative assessment

Task	% of module mark
Closed/in-person Exam (Centrally scheduled)	100

Special assessment rules

None

Indicative reassessment

Task	% of module mark
Closed/in-person Exam (Centrally scheduled)	100

Module feedback

Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines.

Indicative reading

** Forsyth and Ponce Computer Vision a Modern Approach Prentice Hall

** Anil K. Jain Fundamentals of Digital Image Processing Prentice Hall

CRYPTOGRAPHY THEORY & PRACTICE

Module summary

The module aims to provide a broad overview of modern cryptography. The module will cover the fundamental security goals achieved through cryptographic algorithms and protocols, how they are formalised, designs that enable achieving those goals, and how formal security arguments can be made for the design achieving the security goals.

Elective Pre-Requisites

These pre-requisites only apply to students taking this module as an elective.

Module will run

Occurrence Teaching period

Module aims

The module aims to provide a broad overview of modern cryptography. The module will cover the fundamental security goals achieved through cryptographic algorithms and protocols, how they are formalised, designs that enable achieving those goals, and how formal security arguments can be made for the design achieving the security goals.

Module learning outcomes

By the end of the module the students will be able to:

describe and apply the fundamental security properties provided by cryptographic algorithms and protocols;

describe how these security properties are formalised;

describe the implications of cryptographic security arguments;

identify security requirements for a given practical scenario and identify the appropriate cryptographic tools to achieve the security requirements;

assess whether given cryptographic algorithms and protocols meet identified security requirements; and

analyse the security and efficiency of cryptographic protocols from both theoretical and practical points of view.

Indicative assessment

Task % of module mark

Closed/in-person Exam (Centrally scheduled) 100

Special assessment rules

None

Indicative reassessment

Task	% of module mark
Closed/in-person Exam (Centrally scheduled)	100

Module feedback

Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines.

Indicative reading

[1] J. Katz, Introduction to modern cryptography, Third edition :: Chapman & Hall/CRC, 2020
[2] J.-P. Aumasson, Serious cryptography : a practical introduction to modern encryption :: No Starch Press, 2018
[3] C. Paar, Understanding cryptography : a textbook for students and practitioners :: Springer, 2009
[4] A. McAndrew, Introduction to cryptography with open-source software :: CRC Press, 2011
[5] W. Stallings, Cryptography and network security : principles and practice, Seventh edition :: Pearson, 2017
[6] A. J. Menezes, Handbook of applied cryptography :: CRC Press, 1997

EMBEDDED SYSTEMS DESIGN & IMPLEMENTATION

Module will run

Occurrence Teaching period

Module aims

The aim of this module is to understand the need for embedded computer systems, and the engineering process to design, implement and validate them.

Module learning outcomes

Articulate the need for embedded computer systems, and the requirements imposed on them by their application scenario.

Be able to apply an engineering process to design, implement and validate embedded systems.

Demonstrate the different levels of abstraction that are used throughout the design process, and be able to decide the most appropriate abstractions at each step.

Explore hardware/software implementation trade-offs, and a number of partitioning, mapping and evaluation techniques that can be used to analyse that trade-off for a particular application scenario.

Program computing platforms that have limited performance, energy, memory and storage capacity.

Design and evaluate custom hardware architectures.

Understand the process to improve and assure timing properties, including priority assignment, scheduling of real-time tasks and schedulability analysis.

Indicative assessment

Task	% of module mark
Essay/coursework	100

Special assessment rules

None

Indicative reassessment

Task	% of module mark
Essay/coursework	100

Module feedback

Feedback is given to each student on each of the two assessments, highlighting the strengths and weaknesses of the proposed design. Additional feedback is also given during practical sessions, which are also based on design problems.

Indicative reading

P. Marwedel, Embedded System Design, Springer, 2011

ENGINEERING 2: AUTOMATED SOFTWARE ENGINEERING

Module will run

Occurrence Teaching period

Module aims

This module builds on ENG1 and introduces advanced techniques for engineering large, data-intensive software systems that consist of components written in different programming languages and operating on a variety of platforms. In order to manage the complexity of engineering such data-intensive systems, ENG2 will introduce the theory, principles and practices of model-driven engineering (MDE), focusing on technical topics. These include modelling, metamodelling, model management, model transformation, model-driven traceability, workflows, model validation, and model evolution. Non-technical issues including standards, domain-specific MDE versus general-purpose MDE and MDE processes will also be considered. The discussion of concepts and technologies for engineering modern data-intensive software systems will start from a general discussion about architectural concerns, then move on to discuss their design, implementation as a collaboration of microservices, and using containers for reproducible and scalable deployment. ENG2 will discuss assurance practices around data-intensive software systems, such as testing and hardening.

Module learning outcomes

Demonstrate the principles and practices of modelling and metamodelling.

Implement and orchestrate domain-specific models, metamodels and model management operations.

Demonstrate the design principles for data-intensive software systems, including their architecture characteristics like availability, scalability, reliability and security, and the characteristics of the organisation.

Structure data-intensive systems as combinations of independently maintained microservices, and propose benefits and challenges in terms of complexity management, organisational scalability, and resource usage.

Deploy the components of data-intensive systems in a scalable and reproducible manner, by packaging them as containers and orchestrating these containers to work with each other.

Describe assurance practices in the development and deployment of large data-intensive systems, including testing and hardening.

Articulate how Model-Driven Engineering integrates with wider system engineering processes and policies, including those of a data-intensive software system.

Indicative assessment

Task % of module mark

Essay/coursework 100

Special assessment rules
None

Indicative reassessment
Task % of module mark
Essay/coursework 100
Module feedback
Feedback is provided throughout the sessions, and after the assessment as per normal University guidelines.

Indicative reading
Newman, Sam. Building Microservices, O'Reilly Media, 2021.
Kleppmann, Martin. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, O'Reilly Media, 2017.
Wasowski A. Domain-Specific Languages: Effective Modeling, Automation, and Reuse. 1st ed. 2023. (Berger T, ed.). Springer International Publishing; Imprint Springer; 2023.

ETHICAL HACKING, ANALYSIS & INVESTIGATION

Module aims

The module aims to provide an introductory range of theoretical and practical skills to undertake ethical hacking, analysis, and investigation of modern computer systems and networks.

The module will explore ethical hacking topics such as vulnerability exploitation of cryptographic and network protocols, reverse engineering, and other penetration testing techniques, as well as malware analysis and digital investigation of cyber incidents.

The module will familiarise students with a range of tools used by ethical hackers and security practitioners.

Module learning outcomes

By the end of the module the students will be able to:

Explain the fundamental concepts of cyber security in systems and networks

Understand the security strengths & weaknesses in network mechanisms

Understand the major threats and attacks in systems and networks under various scenarios, architectures, and threat models

Analyse control solutions for network security

Assess the relative merits of different solution approaches in various security-related contexts

Indicative assessment

Task	% of module mark
Closed/in-person Exam (Centrally scheduled)	50
Closed/in-person Exam (Centrally scheduled)	50

Special assessment rules

None

Indicative reassessment

Task	% of module mark
Closed/in-person Exam (Centrally scheduled)	50
Closed/in-person Exam (Centrally scheduled)	50

Module feedback

Feedback is provided throughout the sessions, and after the assessment as per normal University guidelines.

Indicative reading

Jon Erickson. Hacking: The Art of Exploitation. No Starch (2008)

Sikorski, Michael ; Honig, Andrew. Practical Malware Analysis: A Hands-On Guide to Dissecting Malicious Software. No Starch Press, Incorporated (2012)

HIGH-INTEGRITY SYSTEMS ENGINEERING

Module summary

This module teaches students to adapt their software development practice to take account of the general criticality and the specific risks of the software they are developing.

Related modules

Students on the joint Mathematics and Computer Science programmes who are interested in this module should discuss the necessary prerequisite knowledge with the module leader.

Elective Pre-Requisites

These pre-requisites only apply to students taking this module as an elective.

Good knowledge of software engineering practice.

Module will run

Occurrence Teaching period

Module aims

This module teaches students to adapt their software development practice to take account of the general criticality and the specific risks of the software they are developing. Students will gain a broad understanding of the issues involved in designing and implementing critical systems, be aware of the methods used to construct critical systems, and understand the limitations of the various methods, analysis techniques and tools currently in use.

Module learning outcomes

Understand and articulate the issues involved in designing and implementing critical systems

Apply a range of techniques to the design, and validation of high integrity systems, such as fault-tree analysis and failure analysis.

Discuss issues of high integrity engineering, both technical and social.

Participate in significant discussion periods brainstorming scenarios and discussing previous well-documented examples of system failures

Introduce existing software engineering concepts for the development of critical systems

Indicative assessment

Task	% of module mark
Online Exam -less than 24hrs (Centrally scheduled)	100

Special assessment rules

None

Indicative reassessment

Task	% of module mark

Online Exam -less than 24hrs (Centrally scheduled) 100

Module feedback

Feedback is provided through work in practical sessions, through discussion sessions, and after the final assessment as per normal University guidelines.

Indicative reading

N.G. Leveson, Safeware: System Safety and Computers, Addison-Wesley, 1995

HIGH-PERFORMANCE PARALLEL & DISTRIBUTED SYSTEMS

Module summary

This module introduces and explores the use of high-performance computing, and related technologies.

Module aims

This module introduces and explores the use of HPC and related technologies, covering on and off node parallelism, accelerators, and memory management. Students will understand the increasingly important role that HPC plays in science and engineering. Throughout the module, students will practice by developing programs using a range of parallel programming paradigms like OpenMP and MPI.

Module learning outcomes

Demonstrate the development of parallel programs on shared memory systems using popular programming paradigms.

Be able to understand appropriate metrics to assess the performance of applications and systems.

Demonstrate the development of parallel programs on distributed memory systems using appropriate parallel programming libraries.

Be able to explore the design space afforded by parallel and distributed systems, including issues of GPU programming, high-speed networking, and I/O.

Be able to reason about the performance, portability, and productivity of different approaches to developing HPC applications.

Indicative assessment

Task	% of module mark
Essay/coursework	100

Special assessment rules

None

Indicative reassessment

Task	% of module mark
Essay/coursework	100

Module feedback

Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines.

Indicative reading

Introduction to High Performance Scientific Computing, Eijkhout, Victor; van de Geijn, Robert; Chow, Edmond

HUMAN FACTORS: TECHNOLOGY IN CONTEXT

Additional information

Pre-Requisite Module:

Human-Computer Interaction (HCIN) - COM00018C

Pre-Requisite Knowledge:

Research Methods

Elective Pre-Requisites
These pre-requisites only apply to students taking this module as an elective.

None.
Module will run
Occurrence Teaching period
Module aims
This module will equip students with an understanding of how human characteristics and design influence interactions with technical systems in organisational and industry contexts. The module uses real-world case studies to introduce and critically evaluate the contribution of humans in developing and maintaining safe, ethical and secure systems. The module aims to develop students' skills in:

Analysis techniques

Proposing and applying appropriate criteria to assess the rigour and suitability of industrial and academic approaches to human reliability.

Module learning outcomes
Recognise the role of human characteristics/factors/mental models in relation to the safe, secure and effective operation and maintenance of interactive systems across applied/industry contexts

Critically evaluate the quality of designs and propose solutions to security and safety issues

Demonstrate the ability to locate and, where relevant, apply existing academic research to inform the consideration of issues arising within human-technology interactions

Identify, apply and critique appropriate techniques for human reliability and error analysis

Identify, apply and critique appropriate techniques for security analysis, including issues of authentication, privacy and foreseeable misuse

Use appropriate frameworks and design techniques to investigate ethical issues in relation to human-machine interaction

Indicative assessment

Task	% of module mark
Essay/coursework	100

Special assessment rules
None

Indicative reassessment

Task	% of module mark
Essay/coursework	100

Module feedback
Feedback is provided throughout the sessions, and after the assessment as per normal University guidelines.

INTELLIGENT SYSTEMS: PROBABILISTIC & DEEP LEARNING
Additional information
Pre-Requisite knowledge - Understanding of the theory and practice of Machine Learning. For undergraduates this is covered in intelligent systems modules (for example, INT2 - Intelligent Systems 2: Machine Learning & Optimisation COM00024I or IMLO - Intelligent Systems: Machine Learning & Optimisation COM00026I).

Elective Pre-Requisites
These pre-requisites only apply to students taking this module as an elective.

A Level Maths or Equivalent plus good understanding of the theory and practice of machine learning.

Module will run
Occurrence Teaching period
Module aims
This module builds on the basic machine learning covered previously in the programme, and takes students up to state of the art methods in modern deep learning. It introduces probabilistic methods, where we can reason about uncertainty, and deep learning based methods, where neural networks with many layers prove to be the most powerful general model for learning. We will see a range of methods and architectures for classification and regression problems, unsupervised generative models and the mathematics that underlies these techniques. We will cover both theory and practicalities: how are these ideas actually implemented in a modern machine learning library like PyTorch?

Module learning outcomes
Explain the probabilistic basis of machine learning

Demonstrate a working knowledge of manifold embedding and kernel methods

Apply a range of Bayesian methods for regression, classification and clustering

Be familiar with the main deep learning architectures

Demonstrate the optimisation process and different variants (i.e. gradient descent, stochastic algorithms, ADAM)

Indicative assessment
Task % of module mark
Essay/coursework 100
Special assessment rules
None

Indicative reassessment
Task % of module mark
Essay/coursework 100
Module feedback
Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines

Indicative reading
Solomon, Justin. Numerical Algorithms. AK Peters/CRC Press, 2015

LEGAL PRACTICE, TECHNOLOGY & COMPUTER SCIENCE

Module summary

Law and computer science students will work collaboratively in teams to develop a technology-based solution to a real-life legal practice process problem. From initial analysis of the problem, teams will work to develop a solution to the problem, assessing both legal and procedural issues, system and user design requirements, functionality, costs, benefits and risks. Students from each discipline will contribute know-how from their discipline to the project, whilst gaining new understanding and skills. The module will be supported by a leading international law firm with a technology hub in the region, providing students with access to external expertise in the field, in addition to disciplinary tutor support.

Module aims

The aim of the module is to provide law and computer science students with an opportunity to develop an applied understanding of how technology and computer science are being applied in developing new approaches to the provision of legal services and how these can offer greater accessibility to justice, as well as efficiency, quality and costs gains. Students will develop knowledge both within their own and the colloborative discipline, as well as a range of analytical, problem-solving, planning, communication and interpersonal skills. The module also aims to provide students with opportunities to interact with experts in legal practice technology and thus gain contemporary professional perspectives on the areas covered in the module.

Module learning outcomes

On completion of this module, students should be able to:

Analyse a legal process which may be improved for users by application of computer science
Apply design thinking to identify procedural, technical, legal and user issues
Communicate orally and in writing relevant principles from their primary discipline to colleagues from another discipline
Collaborate and synthesise disciplinary principles to develop potential solutions
Evaluate potential solutions against user requirements
Formulate a costed implementation plan for an agreed solution
Present a persuasive written and oral case for the solution
Explain the potential applications of computer science and technology in the development of legal services
Reflect on learning gains and challenges from the module, including cross-disciplinary collaboration

Module content

Law and computer science students will work collaboratively to develop a solution to a legal process problem, based on a real-life access to justice scenario. This will be developed in collaboration with a law firm.

The module will use problem-based learning (PBL) techniques, and be predominantly group-assessed. Students will receive a problem and, working in teams comprising equal numbers from each discipline will apply PBL techniques to identify:

substantive and procedural issues
client and internal commercial issues
risks
process requirements
Using these as the basis to develop a solution, there will be an element of cross-discipline teaching: law students will have to be able to explain law, procedure and process requirements to computer science students and the latter will have to explain tech functionality and capabilities, in each case in language understandable to those from the other discipline.

Following initial detailed analysis of the problem, teams will work to develop a solution to the problem, with activity moving from analysis to development. They will plan a programme of work outside class activity sessions, with the latter acting as formal workshops/surgeries, during which teams can obtain feedback from facilitators from both disciplines. There will also be an opportunity to obtain feedback from a legal technology expert from a law firm, as part of a plenary "masterclass".

The assessment will require submission of the solution - e.g., system requirements; functionality; process map; costs; benefits; risk; time-line to implementation - in the form of a business proposal, together with an oral presentation by each group. Individual students will also submit a personal reflection on learning gained against the module outcomes.

Indicative assessment
Task % of module mark
Essay/coursework 20
Groupwork 50
Oral presentation/seminar/exam 30
Special assessment rules
None

Additional assessment information
Formative feedback will be provided on a rolling basis by facilitators as teams progress through the analysis and development stages of the module, especially during workshops.

Indicative reassessment
Task % of module mark
Essay/coursework 20
Groupwork 50
Oral presentation/seminar/exam 30

Module feedback
Students will receive summative feedback as follows:

oral feedback on group presentation
written feedback on group submission
written feedback on individual reflective submission
Indicative reading
The Future of the Professions: Susskind & Susskind – OUP 2017

Tomorrow's Lawyers – Susskind – OUP 2017

The End of Lawyers? Susskind – OUP 2008

NETWORK SECURITY

Module summary

Network Security (NETS): This module covers the basic concepts of cyber security, how these are modelled, threat models, and the mechanisms to enforce security policies.

Elective Pre-Requisites

These pre-requisites only apply to students taking this module as an elective.

A Level Maths or Equivalent.

Module will run

Occurrence Teaching period

Module aims

This module aims to provide a broad knowledge of network and system security, addressing threats over a range of network layers and detailing corresponding defensive countermeasures and protocols. The module will cover the basic concepts of cyber security (confidentiality, integrity and availability), how these are modelled, threat models (adversary capabilities and goals), and basic control mechanisms to enforce security policies (e.g. access control). Students will learn to understand network security, threats, and the mechanisms that have been developed to counter them. It explores a range of different networked systems, the main network attacks, and their defence mechanisms.

Module learning outcomes

By the end of the module the students will be able to:

Describe the fundamental concepts of cyber security in systems and networks
Identify security strengths & weaknesses in network mechanisms
Identify major threats and attacks in systems and networks under various scenarios, architectures, and threat models
Propose control solutions for network security
Assess the relative merits of different solution approaches in various security-related contexts

Indicative assessment

Task % of module mark
Essay/coursework 100

Special assessment rules

None

Indicative reassessment

Task % of module mark
Essay/coursework 100

Module feedback

Feedback is provided through work in practical sessions, discussion in seminars, and after each assessment as per normal University guidelines.

Indicative reading

Sherri Davidoff, Jonathan Ham, Network Forensics: Tracking Hackers Through Cyberspace, Prentice Hall, 2012

Kevin R Fall, W Richard Stevens, TCP/IP Illustrated, Volume 1: The protocols, Addison Wesley, 2012

Andrew Tannenbaum, Computer Networks, Prentice Hall, 2002

PLAYER EXPERIENCES IN DIGITAL GAMES

Module aims

This module will provide students with a comprehensive understanding of player experience and the different ways in which games can impact players. In addition to learning about what player experience is and how to evaluate it, the module will also cover the effects of games on players (e.g. in relation to wellbeing) and the use of games for applied purposes (e.g. behaviour change).

Module learning outcomes

Demonstrate an understanding of the breadth of player experience

Select appropriate methods for evaluating different kinds of player experience

Conduct playtesting sessions and report on the findings

Compare and contrast the different ways in which digital games can affect players

Plan appropriate evaluations for applied games

Indicative assessment

Task	% of module mark
Essay/coursework	60
Groupwork	40

Special assessment rules

None

Indicative reassessment

Task	% of module mark
Essay/coursework	60
Groupwork	40

Module feedback

Feedback is provided throughout the sessions, and after the assessment as per normal University guidelines.

Indicative reading

Drachen, A., Mirza-Babaei, P., & Nacke, L. E. (Eds.). (2018). Games user research. Oxford University Press.

Stahlke, S., & Mirza-Babaei, P. (2022). The Game Designer's Playbook: An Introduction to Game Interaction Design. Oxford University Press.

QUALITATIVE APPROACHES TO INVESTIGATING UX

Module aims

This module will provide students the methodological approaches to interrogating concepts and users' subjective experiences with interactive systems. The module provides theoretical and practical grounding in methods used for collection and analysis of qualitative data that are used in industry and academic research. Students will learn how different qualitative methods can be used for a range of purposes, from informing to design to understanding technology use in context.

Module learning outcomes

Identify and justify use of appropriate qualitative methodology

Recognise and acknowledge the underpinning epistemological positions of qualitative methods

Recognise the role of the researcher in the research process

Critique and contrast the methods of studies against the standards of rigour and validity of the chosen methodology

Report and discuss the results of an analysis appropriate to the method used

Describe the important aspects of research governance including ethical conduct and data governance

Indicative assessment

Task	% of module mark
Essay/coursework	100

Special assessment rules

None

Indicative reassessment

Task	% of module mark
Essay/coursework	100

Module feedback

Feedback is provided throughout the sessions, and after the assessment as per normal University guidelines.

QUANTUM COMPUTATION

Module summary

The aim of this module is to introduce the theory of quantum computation. In it we will learn about the pioneering quantum algorithms that promise a qualitative leap in computation power over conventional computers.

Elective Pre-Requisites

These pre-requisites only apply to students taking this module as an elective.

A Level Maths or Equivalent; good understanding of programming.

Module will run

Occurrence Teaching period

Module aims

Introducing both the promise and limitations of quantum computation. Gate operations, evolving quantum states, calculating the result of measurements on quantum states, designing and analyzing quantum computational circuits, key algorithms (e.g., Shor's, Grover's and the Deutsch-Jozsa algorithms).

Module learning outcomes

Articulate both the promise and limitations of quantum computation.

Apply some of the many concepts and techniques in quantum computation (e.g., applying gate operations and evolving quantum states, calculating the result of measurements on quantum states, designing and analyzing quantum computational circuits);

Explain some of the key algorithms (e.g., Shor's, Grover's and the Deutsch-Jozsa algorithms) and their implications, and are able to simulate these algorithms on quantum states.

Indicative assessment

Task	% of module mark
Online Exam -less than 24hrs (Centrally scheduled)	100

Special assessment rules

None

Indicative reassessment

Task	% of module mark
Online Exam -less than 24hrs (Centrally scheduled)	100

Module feedback

Feedback is provided through work in practical sessions, and after the final assessment as per normal University guidelines.

Indicative reading

*** G. Benenti, et al., Principles of Quantum Computation and Information, vol I, World Scientific, 2004

*** G. Benenti, et al., Principles of Quantum Computation and Information, vol II, World Scientific, 2007

*** P. Kaye, et al., An Introduction to Quantum Computing, Oxford University Press, 2007

RESEARCH METHODS IN COMPUTER SCIENCE

Module summary

A core module that equips postgraduate taught students with core theorectical and practical research skills.

Module will run

Occurrence Teaching period

Module aims

Research is about both generating new knowledge and evaluating confidence in knowledge. There are three skills associated with the conduct of good research:

The ability to accurately identify from existing literature and systems a meaningful or important gap in knowledge, and therefore what constitutes new knowledge in the domain of Computer Science

The ability to competently intervene in the world (e.g. through developing systems, implementing data collection procedures, conducting experiments…) in order to generate knowledge that causes positive change.

The ability to evaluate the quality of evidence stemming from an intervention using sound analysis, and communicate that analysis to the scientific community.

As the ways of intervening in the world depend strongly on your disciplinary area, the aim of this module is to provide an introduction to the first and third pillars, namely the conduct of literature reviews to identify a research gap; and the methods for analysing research data, interpreting said analysis, and accurately presenting research outcomes.

The module will also cover general principles related to the second pillar that apply across disciplinary areas in the development of novel methods and conduct of experiments.

Module learning outcomes

By the end of this module, students will be able to…

Identify a gap in the evidence base within the structure of a formal academic literature review.

Discuss the validity and reliability of methods used in extant or novel research.

Describe and apply principles of responsible research and innovation to a research project.

Analyse a variety of quantitative research data using an array of appropriate inferential statistical tests.

Present the outcomes of a research project.

Indicative assessment

Task % of module mark
Essay/coursework 50
Groupwork 50
Special assessment rules
None

Indicative reassessment
Task % of module mark
Essay/coursework 50
Essay/coursework 50
Module feedback
Feedback is provided throughout the sessions, and after the assessment as per normal University guidelines.

Indicative reading
Howell DC. Fundamental Statistics for the Behavioral Sciences . 9th edition, student edition. Cengage Learning; 2017.
Goldbort R. Writing for Science . Yale University Press; 2006.