National 5 Course Specification

SQA

National 5 Computing Science

Course code:	C816 75
Course assessment code:	X816 75
SCQF:	level 5 (24 SCQF credit points)
Valid from:	session 2023–24

The course specification provides detailed information about the course and course assessment to ensure consistent and transparent assessment year on year. It describes the structure of the course and the course assessment in terms of the skills, knowledge and understanding that are assessed.

This document is for teachers and lecturers and contains all the mandatory information you need to deliver the course.

This edition: May 2023 (version 3.0)

Contents

Course overview

The course consists of 24 SCQF credit points which includes time for preparation for course assessment. The notional length of time for a candidate to complete the course is 160 hours.

The course assessment has two components.

Component	Marks	Duration
Question paper	80	1 hour and 30 minutes
Assignment	40	See course assessment section

Recommended entry	Progression
Entry to this course is at the discretion of the centre. Candidates should have achieved the fourth curriculum level or the National 4 Computing Science course or equivalent qualifications and/or experience prior to starting this course.	♦ other qualifications in computing science or related areas ♦ further study, employment and/or training

Conditions of award

The grade awarded is based on the total marks achieved across all course assessment components.

Achievement of this course gives automatic certification of the following Core Skill:

♦ Information and Communication Technology at SCQF level 5

Course rationale

National Courses reflect Curriculum for Excellence values, purposes and principles. They offer flexibility, provide more time for learning, more focus on skills and applying learning, and scope for personalisation and choice.

Every course provides opportunities for candidates to develop breadth, challenge and application. The focus and balance of assessment is tailored to each subject area.

The National 5 Computing Science course encourages candidates to become successful, responsible and creative in using technologies, and to develop a range of qualities including flexibility, perseverance, confidence, and enterprise.

At this level, the course covers a common core of concepts which underpin the study of computing science and explores the role and impact of contemporary computing technologies. It also includes a range of transferable skills, which opens up a wide range of career and study opportunities.

Purpose and aims

The course helps candidates to understand computational processes and thinking. It covers a number of unifying themes that are used to explore a variety of specialist areas, through practical and investigative tasks.

The course highlights how computing professionals are problem-solvers and designers, and the far-reaching impact of information technology on our environment and society.

It enables candidates to:

♦ apply computational-thinking skills across a range of contemporary contexts
♦ apply knowledge and understanding of key concepts and processes in computing science
♦ apply skills and knowledge in analysis, design, implementation, testing and evaluation to a range of digital solutions
♦ communicate computing concepts and explain computational behaviour clearly and concisely using appropriate terminology
♦ develop an understanding of the role and impact of computing science in changing and influencing our environment and society

Who is this course for?

This course is designed for learners who are considering further study or a career in computing science and related disciplines. It provides opportunities to enhance skills in planning and organising, working independently and in teams, critical thinking and decision making, research, communication, and self- and peer-evaluation, in a range of contexts.

Course content

The course has four areas of study:

Software design and development

Candidates develop knowledge, understanding and practical problem-solving skills in software design and development, through a range of practical and investigative tasks using appropriate software development environments. This develops their programming and computational-thinking skills by implementing practical solutions and explaining how these programs work. Tasks involve some complex features (in both familiar and new contexts), that require some interpretation by candidates. They are expected to analyse problems, and design, implement, test and evaluate their solutions.

Computer systems

Candidates develop an understanding of how data and instructions are stored in binary form and basic computer architecture. They gain an awareness of the environmental impact of the energy use of computing systems and security precautions that can be taken to protect computer systems.

Database design and development

Candidates develop knowledge, understanding and practical problem-solving skills in database design and development, through a range of practical and investigative tasks. This allows candidates to apply computational-thinking skills to analyse, design, implement, test, and evaluate practical solutions, using a range of development tools such as SQL. Tasks involve some complex features (in both familiar and new contexts), that require some interpretation by candidates.

Web design and development

Candidates develop knowledge, understanding and practical problem-solving skills in web design and development, through a range of practical and investigative tasks. This allows candidates to apply computational-thinking skills to analyse, design, implement, test and evaluate practical solutions to web-based problems, using a range of development tools such as HTML, CSS and Javascript. Tasks involve some complex features (in both familiar and new contexts), that require some interpretation by candidates.

Skills, knowledge and understanding

Skills, knowledge and understanding for the course

The following provides a broad overview of the subject skills, knowledge and understanding developed in the course:

♦ applying aspects of computational thinking across a range of contexts
♦ analysing problems within computing science across a range of contemporary contexts

- ♦ designing, implementing, testing and evaluating digital solutions (including computer programs) to problems across a range of contemporary contexts
- ♦ developing skills in computer programming and the ability to communicate how a program works, by being able to read and interpret code
- ♦ communicating understanding of key concepts related to computing science, clearly and concisely, using appropriate terminology
- ♦ understanding of legal implications and environmental impact of contemporary technologies
- ♦ applying computing science concepts and techniques to create solutions across a range of contexts

Skills, knowledge and understanding for the course assessment

The following provides details of skills, knowledge and understanding sampled in the course assessment:

Software design and development	
Development methodologies	Describe and implement the phases of an iterative development process: analysis, design, implementation, testing, documentation, and evaluation, within general programming problem-solving.
Analysis	Identify the purpose and functional requirements of a problem that relates to the design and implementation at this level, in terms of: ♦ inputs ♦ processes ♦ outputs
Design	Identify the data types and structures required for a problem that relates to the implementation at this level, as listed below. Describe, identify, and be able to read and understand: ♦ structure diagrams ♦ flowcharts ♦ pseudocode Exemplify and implement **one** of the above design techniques to design efficient solutions to a problem. Describe, exemplify, and implement user-interface design, in terms of input and output, using a wireframe.
Implementation (data types and structures)	Describe, exemplify, and implement appropriately the following data types and structures: ♦ character ♦ string

Software design and development	
	♦ numeric (integer and real)
	♦ Boolean
	♦ 1-D arrays

Implementation (computational constructs)	Describe, exemplify, and implement the appropriate constructs in a high-level (textual) language:
	
	♦ expressions to assign values
	♦ expressions to return values using arithmetic operations (addition, subtraction, multiplication, division, and exponentiation)
	♦ expressions to concatenate strings
	♦ selection constructs using simple conditional statements with <, >, ≤, ≥, =, ≠ operators
	♦ selection constructs using complex conditional statements
	♦ logical operators (AND, OR, NOT)
	♦ iteration and repetition using fixed and conditional loops
	♦ predefined functions (with parameters):
	  — random
	  — round
	  — length
	
	Read and explain code that makes use of the above constructs.

Implementation (algorithm specification)	Describe, exemplify, and implement standard algorithms: ♦ input validation ♦ running total within loop ♦ traversing a 1-D array
Testing	Describe, identify, exemplify, and implement normal, extreme, and exceptional test data for a specific problem, using a test table. Describe and identify syntax, execution, and logic errors.

Evaluation	Describe, identify, and exemplify the evaluation of a solution in terms of:
	
	♦ fitness for purpose
	♦ efficient use of coding constructs
	♦ robustness
	♦ readability:
	  — internal commentary
	  — meaningful identifiers
	  — indentation
	  — white space

Computer systems	
Data representation	Describe and exemplify the use of binary to represent positive integers.
	Describe floating point representation of positive real numbers using the terms mantissa and exponent.
	Convert from binary to denary and vice-versa.
	Describe extended ASCII code (8-bit) used to represent characters.
	Describe the vector graphics method of graphic representation for common objects:
	♦ rectangle ♦ ellipse ♦ line ♦ polygon
	with attributes:
	♦ co-ordinates ♦ fill colour ♦ line colour
	Describe the bit-mapped method of graphics representation.
Computer structure	Describe the purpose of the basic computer architecture components and how they are linked together:
	♦ processor (registers, ALU, control unit) ♦ memory locations with unique addresses ♦ buses (data and address)
	Explain the need for interpreters and compilers to translate high-level program code to binary (machine code instructions).
Environmental impact	Describe the energy use of computer systems, the implications on the environment and how these could be reduced through:
	♦ settings on monitors ♦ power down settings ♦ leaving computers on standby
Security precautions	Describe the role of firewalls.
	Describe the use made of encryption in electronic communications.

Database design and development	
Analysis	Identify the end-user and functional requirements of a database problem that relates to the implementation at this level.

Design	Describe and identify the implications for individuals and businesses of the UK General Data Protection Regulation (UK GDPR) that data must be:
	
	♦ processed lawfully, fairly and in a transparent manner in relation to individuals
	♦ used for the declared purpose only
	♦ limited to the data needed for the declared purpose
	♦ accurate
	♦ not kept for longer than necessary
	♦ held securely
	
	Describe and exemplify entity-relationship diagrams with two entities indicating:
	
	♦ entity name
	♦ attributes
	♦ relationship (one-to-many)
	
	Describe and exemplify a data dictionary:
	
	♦ entity name
	♦ attribute name
	♦ primary and foreign key
	♦ attribute type:
	  — text
	  — number
	  — date
	  — time
	  — Boolean
	♦ attribute size
	♦ validation:
	  — presence check
	  — restricted choice
	  — field length
	  — range
	
	Exemplify a design of a solution to the query:
	
	♦ multiple tables

Database design and development	
	♦ fields ♦ search criteria ♦ sort order

Implementation	Implement relational databases with two linked tables, to match the design with referential integrity.
	
	Describe, exemplify and implement SQL operations for pre-populated relational databases, with a maximum of two linked tables:
	
	♦ select:
	  — from
	  — where:
	    ○ AND, OR, <, >, =
	    ○ order by with a maximum of two fields
	♦ insert
	♦ update
	♦ delete
	♦ equi-join between tables
	
	Read and explain code that makes use of the above SQL.

Testing	Describe and exemplify testing: ♦ SQL operations work correctly at this level
Evaluation	Evaluate solution in terms of: ♦ fitness for purpose ♦ accuracy of output

Web design and development	
Analysis	Identify the end-user and functional requirements of a website problem that relates to the design and implementation at this level.
Design	Describe and exemplify the website structure with a home page, a maximum of four linked multimedia pages, and any necessary external links. Describe, exemplify and implement, taking into account end-user requirements, effective user-interface design (visual layout and readability) using wire-framing: ♦ navigational links ♦ consistency across multiple pages ♦ relative vertical positioning of the media displayed ♦ file formats of the media (text, graphics, video, and audio) Describe and identify the implications for individuals and businesses of the Copyright, Designs and Patents Act 1988 relating to: ♦ web content (text, graphics, video, and audio) Compare a range of standard file formats: ♦ audio standard file formats WAV and MP3 in terms of compression, quality, and file size ♦ bit-mapped graphic standard file formats JPEG, GIF, and PNG in terms of compression, animation, transparency, and colour depth Describe the factors affecting file size and quality, relating to resolution, colour depth, and sampling rate. Describe the need for compression. Describe, exemplify and implement prototyping (low-fidelity) from wireframe design at this level.

Implementation (CSS)	Describe, exemplify and implement internal and external Cascading Style Sheets (CSS):
	
	♦ selectors, classes and IDs
	♦ properties
	  — text:
	    ○ font (family, size)
	    ○ color
	    ○ alignment
	  — background colour
	Read and explain code that makes use of the above CSS.

Web design and development	
Implementation (HTML)	Describe, exemplify and implement HTML code: ♦ HTML ♦ head ♦ title ♦ body ♦ heading ♦ paragraph ♦ DIV ♦ link ♦ anchor ♦ IMG ♦ audio ♦ video ♦ lists — ol, ul and li Describe and implement hyperlinks (internal and external), relative and absolute addressing. Read and explain code that makes use of the above HTML.
Implementation (JavaScript)	Describe and identify JavaScript coding related to mouse events: ♦ onmouseover ♦ onmouseout
Testing	Describe and exemplify testing: ♦ matches user-interface design ♦ links and navigation work correctly ♦ media (such as text, graphics, and video) display correctly ♦ consistency
Evaluation	Evaluate solution in terms of: ♦ fitness for purpose

Skills, knowledge and understanding included in the course are appropriate to the SCQF level of the course. The SCQF level descriptors give further information on characteristics and expected performance at each SCQF level (www.scqf.org.uk).

Skills for learning, skills for life and skills for work

This course helps candidates to develop broad, generic skills. These skills are based on [SQA's Skills Framework: Skills for Learning, Skills for Life and Skills for Work](#) and draw from the following main skills areas:

2   Numeracy

2.1   Number processes
2.3   Information handling

4   Employability, enterprise and citizenship

4.2   Information and communication technology (ICT)

5   Thinking skills

5.3   Applying
5.4   Analysing and evaluating

These skills must be built into the course where there are appropriate opportunities and the level should be appropriate to the level of the course.

Further information on building in skills for learning, skills for life and skills for work is given in the course support notes.

Course assessment

Course assessment is based on the information provided in this document.

The course assessment meets the key purposes and aims of the course by addressing:

♦ breadth — drawing on knowledge and skills from across the course
♦ challenge — requiring greater depth or extension of knowledge and/or skills
♦ application — requiring application of knowledge and/or skills in practical or theoretical contexts as appropriate

This enables candidates to apply:

♦ breadth of knowledge from across the course and depth of understanding, to answer appropriately challenging questions in computing science contexts
♦ knowledge and skills developed through the course, to solve appropriately challenging computing science problems in both practical and theoretical contexts

Course assessment structure: question paper

Question paper   80 marks

The question paper gives candidates an opportunity to demonstrate the following skills, knowledge and understanding:

♦ applying aspects of computational thinking, across a range of contexts
♦ analysing problems within computing science, across a range of contemporary contexts
♦ designing, implementing, testing and evaluating digital solutions (including computer programs) to problems, across a range of contemporary contexts
♦ communicating how a program works
♦ communicating key concepts related to computing science clearly and concisely, using appropriate terminology
♦ understanding the legal implications and environmental impact of contemporary technologies
♦ applying computing science concepts and techniques to create solutions, across a range of contexts

The question paper has 80 marks, which is 67% of the overall marks for the course assessment (120 marks).

The question paper has three sections. Section 1 is mandatory, and candidates have the option to complete either section 2 or section 3.

♦ **Section 1**: Software design and development, and Computer systems — 55 marks
♦ **Section 2**: Database design and development — 25 marks
♦ **Section 3**: Web design and development — 25 marks

Each section begins with a number of short, stand-alone questions. These are predominantly 'C' mark questions, presented in a clear and concise way, in a simple and/or familiar context.

This is followed by more challenging, context-based questions with multiple parts. These require a range of responses including restricted and extended response, designing solutions and writing code, and feature both 'C' mark and 'A' mark questions. Some questions are designed to be more challenging and will require candidates to integrate skills, knowledge and understanding, provide detailed descriptions or explanations, and/or analyse, compare, and evaluate.

The questions will:

♦ assess application of understanding, with very few questions requiring direct recall of knowledge
♦ sample across the course in a balanced way
♦ consist of questions set in meaningful contexts, that require candidates to provide some descriptions and explanations

Note: see the 'Preparing for course assessment' section on p27 in the 'Course support notes' for the full range of marks against each area of content and skills.

SQA's standardised reference language

Questions assessing understanding and application of programming skills are expressed using SQA's standardised reference language. Further information can be found in the document '*Reference language for Computing Science question papers*' which can be downloaded from the National 5 Computing Science subject page on SQA's website.

Where candidates are required to answer by writing code, answers may be expressed using any programming language. Candidates are not expected to write code in SQA's standardised reference language. Marks are awarded for demonstrating understanding, not for the correct use of syntax.

Setting, conducting and marking the question paper

The question paper is set and marked by SQA.

It is conducted in centres under conditions specified for external examinations by SQA. Candidates complete the paper in 1 hour and 30 minutes.

Specimen question papers for National 5 courses are published on SQA's website. These illustrate the standard, structure and requirements of the question papers candidates sit. The specimen papers also include marking instructions.

Course assessment structure: assignment

Assignment   40 marks

The assignment gives candidates an opportunity to demonstrate the following skills, knowledge and understanding:

♦ applying aspects of computational thinking across a range of contexts
♦ analysing problems within computing science across a range of contemporary contexts
♦ designing, implementing, testing and evaluating digital solutions (including computer programs) to problems across a range of contemporary contexts
♦ demonstrating skills in computer programming
♦ applying computing science concepts and techniques to create solutions across a range of contexts

The assignment has 40 marks, which is 33% of the overall marks for the course assessment (120 marks).

The assignment has three tasks. Task 1 is mandatory, and candidates have the option to complete either task 2 or task 3.

♦ **Task 1**: Software design and development — 25 marks
♦ **Task 2**: Database design and development — 15 marks
♦ **Task 3**: Web design and development — 15 marks

A proportion of marks are available for the more challenging aspects of each task, where candidates are required to demonstrate problem-solving skills.

Note: see the 'Preparing for course assessment' section on p27 in the 'Course support notes' for the full range of marks against each area of content and skills.

Setting, conducting and marking the assignment

The assignment is:

♦ set by SQA, on an annual basis
♦ conducted under a high degree of supervision and control

Evidence is submitted to SQA for external marking and all marking is quality assured by SQA.

Specimen assessment tasks for National 5 courses are published on SQA's website. These illustrate the standard, structure and requirements of the assessment tasks candidates complete. The specimen assessment tasks also include marking instructions.

Assessment conditions

Time

The assignment must be carried out within 6 hours, starting at an appropriate point in the course and once all content has been delivered. It is not anticipated that this is a continuous 6-hour session but conducted over several shorter sessions.

Supervision, control and authentication

The assignment is conducted under open-book conditions, but supervised to ensure that the work presented is the candidate's own work.

At the end of each session, and upon completion of the assignment, the teacher must ensure that candidate evidence is stored securely.

Resources

Each candidate must have access to a computer system with a high-level (textual) programming language and software that can create, edit and run SQL, HTML and CSS.

The assignment is conducted under open-book conditions, which means candidates are permitted to access resources such as programming manuals, class notes, textbooks and programs they have written throughout the course.

Reasonable assistance

The assignment consists of three independent tasks. They are designed in a way that does not require teachers to provide support to candidates, other than to ensure that they have access to the necessary resources within the centre.

Once the assignment has been completed, it must not be returned to the candidate for further work to improve their mark.

Evidence to be gathered

All candidate evidence (whether created manually or electronically) must be submitted to SQA in a paper-based format. This includes hard copies of program listings, screenshots or similar, as appropriate.

Volume

There is no word count.

Grading

A candidate's overall grade is determined by their performance across the course assessment. The course assessment is graded A–D on the basis of the total mark for all course assessment components.

Grade description for C

For the award of grade C, candidates will typically have demonstrated successful performance in relation to the skills, knowledge and understanding for the course.

Grade description for A

For the award of grade A, candidates will typically have demonstrated a consistently high level of performance in relation to the skills, knowledge and understanding for the course.

Equality and inclusion

This course is designed to be as fair and as accessible as possible with no unnecessary barriers to learning or assessment.

For guidance on assessment arrangements for disabled candidates and/or those with additional support needs, please follow the link to the assessment arrangements web page: www.sqa.org.uk/assessmentarrangements.

Further information

The following reference documents provide useful information and background.

♦ [National 5 Computing Science subject page](#)
♦ [Assessment arrangements web page](#)
♦ [Building the Curriculum 3–5](#)
♦ [Design Principles for National Courses](#)
♦ [SCQF Framework and SCQF level descriptors](#)
♦ [SCQF Handbook](#)
♦ [SQA Skills Framework: Skills for Learning, Skills for Life and Skills for Work](#)
♦ [Educational Research Reports](#)
♦ [SQA e-assessment web page](#)

Appendix: course support notes

Introduction

These support notes are not mandatory. They provide advice and guidance to teachers and lecturers on approaches to delivering the course. They should be read in conjunction with this course specification and the specimen question paper and/or coursework.

Developing skills, knowledge and understanding

This section provides further advice and guidance about skills, knowledge and understanding that could be included in the course. Teachers and lecturers should refer to this course specification for the skills, knowledge and understanding for the course assessment. Course planners have considerable flexibility to select coherent contexts which will stimulate and challenge their candidates, offering both breadth and depth.

The 'approaches to learning and teaching' section provides suggested experiences and activities that teachers and lecturers can build into their delivery, to develop the skills, knowledge and understanding of the course.

Approaches to learning and teaching

Computing Science, like all National Courses, has been developed to reflect Curriculum for Excellence values, purposes and principles.

The approach to learning and teaching developed by individual centres should reflect these principles. Candidates should be encouraged to participate fully in active learning and practical activities by working together, talking, listening, reading or reflecting on a topic, while the teacher or lecturer acts as a facilitator.

An appropriate balance of teaching methodologies should be used in the delivery of the course and a variety of active learning approaches is encouraged, including the following:

Activity-based learning

Whole-class, direct teaching opportunities should be balanced by activity-based learning using practical tasks. An investigative approach is encouraged, with candidates actively involved in developing their skills, knowledge and understanding by investigating a range of real-life and relevant problems and solutions related to areas of study. Learning should be supported by appropriate practical activities, so that skills are developed at the same time as knowledge and understanding.

Group work

Practical activities and investigations lend themselves to group work, and this should be encouraged. Candidates engaged in collaborative group-work strategies can benefit from each other's knowledge, resources and skills, by questioning, investigating, evaluating and

presenting ideas to each another. 'Working as a team' is not specifically identified as one of the skills for learning, skills for life and skills for work for this course and is not assessed. It is, however, a fundamental aspect of working in the IT and related industries, and so should be encouraged and developed by teachers and lecturers.

Problem-based learning

Problem-based learning (PBL) is another strategy which supports candidates' progress through this course. This method can be best used at the end of a topic, where additional challenge is required to ensure candidates are secure in their knowledge and understanding, and to develop their ability to apply knowledge and skills in less-familiar contexts. Learning through PBL develops candidates' problem-solving, decision making, investigative skills, creative thinking, team working and evaluative skills.

Computational thinking

Computational thinking is recognised as important for all candidates — whether they intend to continue with computing science or not. It includes a set of problem-solving-skills and techniques used by software developers to write programs.

There are various ways of defining computational thinking. One useful structure is to group these problem-solving skills and techniques under five broad headings:

♦ **Abstraction:** seeing a problem and its solution at many levels of detail and generalising the information that is necessary. Abstraction allows an idea or a process to be represented in general terms (eg variables), so it can be used to solve other problems that are similar in nature.
♦ **Algorithms**: the ability to develop a step-by-step strategy for solving a problem. Algorithm design is often based on the decomposition of a problem and the identification of patterns that help to solve the problem. In computing science as well as in mathematics, algorithms are often written abstractly, utilising variables in place of specific numbers.
♦ **Decomposition:** breaking down a task so that a process can be clearly explained to another person — or to a computer. Decomposing a problem frequently leads to pattern recognition and generalisation/abstraction, and thus the ability to design an algorithm.
♦ **Pattern recognition**: the ability to notice similarities or common differences that will help make predictions or lead to shortcuts. Pattern recognition is frequently the basis for solving problems and designing algorithms.
♦ **Generalisation**: realising that a solution to one problem can be used to solve a whole range of related problems.

Underpinning all of these concepts is the idea that computers are **deterministic**: they do exactly what you tell them to do and so can be understood.

While computational thinking can be a component of many subjects, computing science is particularly well placed to deliver it. Teachers and lecturers are encouraged to emphasise, exemplify and make these aspects of computational thinking explicit (at an appropriate level),

wherever there are opportunities to do so throughout the teaching and learning of this course.

Using online and outside resources

Stimulating candidates' interest and curiosity should be a prime objective, throughout the teaching of this course. Engaging with outside agencies or industry professionals can greatly enhance the learning process. Online resources, can provide a valuable addition to teaching and learning activities, encouraging research, collation and storage of information, and evaluation of these materials. Using interactive multimedia learning resources, online quizzes, and web-based software can also be used to support teacher-led approaches.

Assessment activities, used to support learning, can be usefully blended with learning activities throughout the course, for example:

♦ sharing learning intentions/success criteria
♦ using assessment information to set learning targets and next steps
♦ adapting teaching and learning activities based on assessment information
♦ providing confidence-boosting feedback on candidates' progress

Where appropriate, self- and peer-assessment techniques should be encouraged.

Meeting the needs of all candidates

Within any National 5 class, each candidate will have individual strengths and areas for improvement.

For example, there could be candidates capable of achieving a higher level in some aspects of the course. Where possible, they should be given the opportunity to do so.

There could also be candidates who are struggling to achieve National 5 level in some aspects of their course, and who would be better suited to a lower level in these areas.

However, where National 5 candidates have studied National 4 in a previous year, it is important to provide them with new and different contexts for learning to keep them motivated.

Learning about Scotland and Scottish culture can enrich candidates' learning experiences and help them to develop the skills for learning, skills for life and skills for work. This should also help prepare them for taking their place in a diverse, inclusive and participative Scotland and beyond. Where there are opportunities to contextualise approaches to learning and teaching to Scottish contexts, teachers and lecturers should consider doing this.

Advice on distribution of time

The notional length of time for candidates to complete the course is 160 hours. They may be expected to contribute some of their own time in addition to the programmed learning time.

The decision on how to distribute time is at the discretion of teachers and lecturers, depending on candidates' prior learning. Time should be allocated for the course assignment (6 hours) and preparation for the question paper.

Suggested learning activities

The sequence of delivery of the four areas of study is at the discretion of teachers and lecturers.

Software design and development

♦ **Analysis**: candidates working in groups could be given a number of problems to analyse and decide the inputs, processes and outputs that are required.

♦ **Design**: teachers and lecturers could get candidates to solve a complex problem using their chosen design technique. They could then discuss the differences between them and whether some are more efficient solutions than others. Teachers and lecturers could present candidates with a variety of completed program designs, and then discuss the inputs/outputs and processes within each design. Candidates could be asked to find errors in completed designs and suggest solutions.

♦ **Implementation**: teachers and lecturers could combine the development of knowledge and understanding of how programs work, as candidates' progress through the practical tasks involved in developing their own programs.

For example, when candidates are selecting and using appropriate constructs and variables when developing their programs, they will need to understand the purpose and function of these constructs and the purpose of variable types. As candidates develop their programming skills by selecting and using appropriate constructs and assigning values to variables, they will also develop a clear understanding of how these constructs and variables work, and what they can be used for. This will enable them to read, interpret and explain the code in their programs.

Candidates could develop an understanding of the key concepts involved in software development before creating their own programs. They could be given a variety of programs and work in groups to find out the following:
— What is the purpose of the programming constructs?
— How do they work?
— What is the purpose of a range of variable types?

Once candidates gain a sound understanding of the purpose of a range of programming constructs, how they function and the purpose of variable types within a program, they should be well placed to develop their own programs in a high-level (textual) language.

Candidates could work in groups writing code from designs given to them in a number of different design techniques.

Candidates could be supplied with a completed design and associated program. They could then be asked to discuss any differences between the design and the implementation.

- **Testing**: teachers and lecturers could give candidates a variety of programs and ask them to create test tables, and get them to test them to check whether they work. It would be useful to give candidates a number of programs that had logic and syntax errors, to show the benefits of testing.
- **Evaluation**: teachers and lecturers could give groups of candidates some completed programs and ask them to evaluate these in terms of efficient use of coding constructs, fitness for purpose and readability.

Computer systems

- **Data representation**: teachers and lecturers could describe how computers store integers, real numbers, text, vector graphics, and bit-mapped graphics. Candidates could undertake different exercises showing the data representation of a variety of numbers, text and graphics.
- **Computer structure**: candidates could research the basic components of a computer system and create a visual representation. Teachers and lecturers could discuss the need for compilers and interpreters, and explain how they are used when they are writing code.
- **Environmental impact**: candidates working in groups could investigate the settings in a control panel and decide whether they are appropriate or whether they could be altered to save more energy.
- **Security precautions**: teachers and lecturers could discuss the importance of the Enigma machine in WWII and how encrypted messages were required for security. They could explain that this works with electronic communications in today's society. Candidates could be asked to research firewalls and present their findings to the class.

Database design and development

- **Analysis**: candidates could be split into groups, with some being database developers and others being clients. The developers would interview the clients and create the end-user and functional requirements for the database problem.
- **Design**: teachers and lecturers could show candidates how to use simple relational databases before they start to teach them the fundamentals of design. Teachers and lecturers could demonstrate to candidates a completed database with sample data showing how the tables are connected using entity-relationship diagrams. A data dictionary could then be explained, including primary and foreign keys, attribute types and size, and the different types of validation. Candidates could be given different types of exercises to create a data dictionary from given data. Teachers and lecturers could discuss the General Data Protection Regulation in relation to a database that they were designing.
- **Implementation**: teachers and lecturers could show candidates how to use simple SQL operations to create simple searches and sorts on one field. Candidates could then undertake a number of exercises to solve problems relating to searching and sorting relational databases using SQL.
- **Testing and evaluation**: teachers and lecturers could give candidates some SQL code and candidates could test them, and evaluate their fitness for purpose and accuracy of output. Teachers and lecturers could supply candidates with an incorrect SQL operation, along with correct expected output, and ask candidates to identify how to correct the SQL statement in order to produce the expected output.

Web design and development

♦ **Analysis**: candidates could be split into groups, with some being web developers and others being clients. The developers would interview the clients and create the end-user and functional requirements for the problem.

♦ **Design**: teachers and lecturers could get candidates to use wire-framing design techniques to design website structures and pages relating to simple web pages that they have shown them. The class could have a discussion about the implications of the Copyright, Design and Patents Act of the website they are designing. When the website designs are completed, groups of candidates could create low-fidelity prototypes. Teachers and lecturers could discuss audio and graphic file types and discuss when each would be the most appropriate to use in different situations. Using appropriate graphics-editing software, candidates could export a bit-mapped graphic using a variety of setting to research factors affecting file size and quality.

♦ **Implementation**: candidates could be given HTML and CSS code with the web pages and asked to explain which parts of the code relate to the web page. Teachers and lecturers could give candidates some low-fidelity prototyping and ask them to implement using HTML and CSS. Teachers and lecturers could provide completed HTML and CSS files and use them to demonstrate the effect of editing the CSS code.

♦ **Testing and evaluation**: teachers and lecturers could give groups of candidates a number of different websites to look at. They could then create test tables that would check that the websites were working correctly and are fit for purpose.

Resources

Suggested resources are summarised below:

♦ internet-enabled computers and a digital projector
♦ access to a high-level (textual) programming language
♦ access to a database that supports execution of SQL statements
♦ web development tools (script enabled browsers)

Centres may find that existing hardware and software within the computing science classroom provides all that is required to deliver the course.

Some suggested specific online resources:

[date accessed September 2017]

♦ **Software design and development**
  www.java.com
  www.python.org
  www.codeacademy.com
  www.programiz.com/python-programming
  www.livecode.com
  www.draw.io

- **Computer systems**

  www.bbc.co.uk/education/subjects

- **Database design and development**

  www.w3schools.com
  www.codeacademy.com
  www.tutorialspoint.com/sql
  www.sqlcourse.com
  Apex.oracle.com/en

- **Web design and development**

  www.w3schools.com
  www.codeacademy.com
  html.net/tutorials
  www.khanacademy.org
  pencil.evolus.vn
  balsamiq.com
  resources.infosecinstitute.com/prototyping
  Goggles.mozilla.org
  Educade.org/teaching_tools/hackasaurus

Some suggested software development environments

There are no restrictions on the choice of software development environment a centre can use; the decision should be based on the suitability of the chosen environment to support the delivery of the mandatory content of the course.

Below is a list of possible examples of software development environments that might be suitable:

- Python
- Live Code
- Visual Basic
- True Basic
- Java
- Xojo (formally real basic)

For additional support, please refer to the 'Resources to support the National 5 Computing Science course' section at the end of this document.

Preparing for course assessment

The course assessment focuses on breadth, challenge and application. Candidates should apply the skills, knowledge and understanding they have gained during the course.

In preparation, candidates should be given opportunities to practise activities similar to those expected in the course assessment. For example, teachers and lecturers could develop questions and tasks similar to those exemplified in the specimen question paper and specimen coursework assessment task.

To assist, we have produced the information below on the following pages:

♦ course assessment overview
♦ question paper brief
♦ assignment brief

Course assessment overview

Marks: 120

The course assessment has two components:

♦ question paper — 80 marks
♦ assignment — 40 marks

Proportion of 'A' and 'C' type marks:

♦ approximately 30% of marks 'A' type
♦ approximately 50% of marks 'C' type

The course assessment (question paper and assignment) is designed using the following ranges of marks against each area of content and skills.

	% of course assessment	Number of marks	Overall	Assignment	Question paper
Analysis	5	6	3-11	3-6	0-5
Design	30	36	28-44	4-8	20-40
Implementation	40	48	40-56	22-24	16-34
Testing	10	12	8-16	3-6	5-10
Evaluation	5	6	3-11	3-6	0-5
Systems	10	12	10-14	n/a	10-14

Note: The skills, knowledge and understanding across the DDD and WDD areas of study are not directly comparable. For example, there is more assessable content for design in DDD than WDD, but more for implementation in WDD than DDD.

As a result, the mark breakdown across analysis, design, implementation, testing and evaluation will not be identical across the options, however, there will be a balance of 'A' type and 'C' type marks across the options in both the question paper and the assignment to ensure a comparable level of demand.

Question paper brief

Marks: 80

Duration: 1 hour and 30 minutes

The question paper has three sections. Section 1 is mandatory, and candidates have the option to complete either section 2 or section 3

♦ **Section 1**: Software design and development, and Computer systems — 55 marks
♦ **Section 2**: Database design and development — 25 marks
♦ **Section 3**: Web design and development — 25 marks

Each section begins with a number of short, stand-alone questions. These are predominantly 'C' mark questions, presented in a clear and concise way, in a simple and/or familiar context.

This is followed by more challenging, context-based questions with multiple parts. These require a range of responses including restricted and extended response, designing solutions and writing code, and feature both 'C' mark and 'A' mark questions.

Proportion of 'A' and 'C' type questions:

♦ approximately 30% of marks 'A' type (primarily in the context-based questions)
♦ approximately 50% of marks 'C' type

The question paper (QP) is designed using the following ranges of marks, against each area of content and skills.

Content	Range of marks
SDD	41-45
CS	10-14
WDD	25
DDD	25

Skills	Range of marks
Analysis	0-5
Design	20-40
Implementation	16-34
Testing	5-10
Evaluation	0-5
Systems	10-14

Note: the marks for skills in the above table are based on the question paper (80 marks). Either combination of SDD/CS and DDD or SDD/CS and WDD falls within these ranges.

The marks for each skill is not identical across the options, for example Implementation for the DDD option may have 28 marks, while the WDD option may have 32 marks.

However, there will be a balance of 'A' type and 'C' type marks across both options to ensure there is a comparable level of demand.

Assignment brief

Marks: 40

Duration: 6 hours

The assignment has three tasks. Task 1 is mandatory, and candidates have the option to complete either task 2 or task 3.

♦ **Task 1**: Software design and development — 25 marks
♦ **Task 2**: Database design and development — 15 marks
♦ **Task 3**: Web design and development — 15 marks

Proportion of 'A' and 'C' type marks:

♦ approximately 30% of marks 'A' type
♦ approximately 50% of marks 'C' type

	Analysis	**Design**	**Implementation**	**Testing**	**Evaluation**	**Total**
SDD	0-5	0-5	15	0-5	0-5	25
DDD	0-5	0-5	7-9	0-5	0-5	15
WDD	0-5	0-5	7-9	0-5	0-5	15
Total	3-6	4-8	22-24	3-6	3-6	**40**

Note: the marks for skills in the above table are based on the assignment (40 marks). Either combination of SDD and DDD task or SDD and WDD falls within these ranges.

The marks for each skill is not identical across the options, however, there will be a balance of 'A' type and 'C' type marks across both options to ensure there is a comparable level of demand.

Developing skills for learning, skills for life and skills for work

Course planners should identify opportunities throughout the course for candidates to develop skills for learning, skills for life and skills for work.

Candidates should be aware of the skills they are developing and teachers and lecturers can provide advice on opportunities to practise and improve them.

SQA does not formally assess skills for learning, skills for life and skills for work.

There may also be opportunities to develop additional skills depending on approaches being used to deliver the course in each centre. This is for individual teachers and lecturers to manage.

Some examples of potential opportunities to practise or improve these skills are provided in the following table:

Skill	How to develop
2 Numeracy	
2.1 Number processing	Candidates could be given opportunities to develop their number processing skills, by practicing problem-solving in numeric-based contexts, eg creating programs that calculate hotel bills or wages.
2.3 Information handling	Information-handling skills could be developed by setting problem-solving contexts where candidates are required to interpret data in different structures, eg flat-files or linked tables in databases, visual layout and navigation for web pages, including appropriate file formats.
4 Employability, enterprise and citizenship	
4.2 Information and communication technology	Throughout the course, candidates should be continuously interacting with the technology around them. This should give plenty of opportunities to extend their ICT skills.
5 Thinking skills	
5.3 Applying	Candidates should be given plenty opportunity to analyse a wide range of problems, apply the knowledge and skills they have acquired in developing information systems, and then test and review their solutions.
5.4 Analysing and evaluating	Candidates should develop skills in analysing and evaluating through the process of creating software solutions to problems.

Resources to support the National 5 Computing Science course

The following table and appendices have been provided as an additional support. Please note that some of these resources are available on external websites that require you to log in or create a user account.

All teaching materials and videos are available on 'Glow', in the folder called 'Revised National 5 Computing Science' within Computing Science Documents. This folder can be accessed from the following link:

glowscotland.sharepoint.com/sites/PLC/technologies/SitePages/Computing%20Science.aspx
[date accessed September 2017]

Software design and development		
Skills, knowledge and understanding		**Exemplification/learning and teaching activities and resources**
Design	Describe, identify, and be able to read and understand: ♦ structure diagrams ♦ flowcharts ♦ pseudocode	See appendix 1 — design techniques www.draw.io
	Exemplify and implement one of the above design techniques to design efficient solutions to a problem.	Teaching design techniques workshop
	Describe, exemplify, and implement user-interface design, in terms of input and output, using a wireframe.	See appendix 2 — user-interface design

Implementation (algorithm specification)	Describe, exemplify, and implement standard algorithms: ♦ input validation ♦ running total within loop ♦ traversing a 1D array	See appendix 3 — standard algorithms

Evaluation	Describe, identify, and exemplify the evaluation of a solution in terms of:
	
	♦ fitness for purpose
	♦ efficient use of coding constructs
	♦ robustness
	♦ readability:
	    — internal commentary
	    — meaningful identifiers
	    — indentation
	    — white space
		See appendix 4 — efficient use of coding constructs

Computer systems		
Skills, knowledge and understanding		**Exemplification/learning and teaching activities and resources**
Data representation	Describe and exemplify the use of binary to represent positive integers.	Lenzie Academy materials
	Describe floating point representation of positive real numbers, using the terms mantissa and exponent.	
	Convert from binary to denary and vice-versa.	
	Describe extended ASCII code (8-bit) used to represent characters.	
	Describe the vector graphics method of graphic representation for common objects:	
	♦ rectangle ♦ ellipse ♦ line ♦ polygon	
	with attributes:	
	♦ co-ordinates ♦ fill colour ♦ line colour	
	Describe the bit-mapped method of graphics representation.	

Computer structure	Describe the purpose of the basic computer architecture components and how they are linked together: ♦ processor (registers, ALU, control unit) ♦ memory locations with unique addresses ♦ buses (data and address) Explain the need for interpreters and compilers to translate high-level program code to binary (machine code instructions).	Lenzie Academy materials
Environmental impact	Describe the energy use of computer systems, the implications on the environment and how these could be reduced through: ♦ settings on monitors ♦ power down settings ♦ leaving computers on standby	Lenzie Academy materials
Security precautions	Describe the role of firewalls. Describe the use made of encryption in electronic communications.	Lenzie Academy materials

Database design and development		
Skills, knowledge and understanding		**Exemplification/learning and teaching activities and resources**
Analysis	Identify the end-user and functional requirements of a database problem that relates to the implementation at this level.	See appendix 14 — database analysis Clydeview Academy materials
Design	Describe and identify the implications for individuals and businesses of the UK General Data Protection Regulation (UK GDPR) that data must be: ♦ processed lawfully, fairly and in a transparent manner in relation to individuals ♦ used for the declared purpose only ♦ limited to the data needed for the declared purpose ♦ accurate ♦ not kept for longer than necessary ♦ held securely	Robert Gordon's College materials
	Describe and exemplify entity-relationship diagrams with two entities indicating: ♦ entity name ♦ attributes ♦ relationship (one-to-many)	See appendix 5 — entity-relationship diagrams DDD delivery video, showing example of teaching databases Clydeview Academy materials Robert Gordon's College materials

	Describe and exemplify a data dictionary:	See appendix 6 — data dictionary
		Clydeview Academy materials
	♦ entity name	
	♦ attribute name	Robert Gordon's College materials
	♦ primary and foreign key	
	♦ attribute type:	
	— text	
	— number	
	— date	
	— time	
	— Boolean	
	♦ attribute size	
	♦ validation:	
	— presence check	
	— restricted choice	
	— field length	
	— range	
	Exemplify a design of a solution to the query:	See appendix 7 — design of solution to database queries
	♦ multiple tables	Clydeview Academy materials
	♦ fields	
	♦ search criteria	
	♦ sort order	

Implementation	Describe, exemplify and implement SQL operations for pre-populated relational databases, with a maximum of two linked tables:
	
	♦ select:
	   — from
	   — where:
	      o AND, OR, <, >, =
	      o order by with a maximum of two fields
	♦ insert
	♦ update
	♦ delete
	♦ equi-join between tables
	
	Read and explain code that makes use of the above SQL.
		www.w3schools.com
		www.sqlcourse.com
		www.codeacademy.com
		www.tutorialspoint.com/sql
		
		Clydeview Academy materials
		
		Robert Gordon's College materials
		
		SQL queries workshop
		
		See appendix 12 — SQL

Testing	Describe and exemplify testing: ♦ SQL operations work correctly at this level	See appendix 8 — testing and evaluation Clydeview Academy materials Robert Gordon's College materials
Evaluation	Evaluate solution in terms of: ♦ fitness for purpose ♦ accuracy of output	Clydeview Academy materials

Web design and development		
Skills, knowledge and understanding		**Exemplification/learning and teaching activities and resources**
Analysis	Identify the end-user and functional requirements of a website problem that relates to the design and implementation at this level.	See appendix 13 — web analysis
Design	Describe and exemplify the website structure with a home page, a maximum of four linked multimedia pages, and any necessary external links.	See appendix 9 — website structure Clydeview Academy materials
	Describe, exemplify and implement, taking into account end-user requirements, effective user-interface design (visual layout and readability) using wire-framing: ♦ navigational links ♦ consistency across multiple pages ♦ relative vertical positioning of the media displayed ♦ file formats of the media (text, graphics, video, and audio)	See appendix 10 — interface design
	Describe, exemplify and implement prototyping (low-fidelity) from wireframe design at this level.	pencil.evolus.vn balsamiq.com resources.infosecinstitute.com/prototyping See appendix 11 — low-fidelity prototyping Clydeview Academy materials

Implementation (CSS)	Describe, exemplify and implement internal and external Cascading Style Sheets (CSS):
	
	♦ selectors, classes and IDs
	♦ properties
	   — text:
	      o font (family, size)
	      o color
	      o alignment
	   — background colour
	
	Read and explain code that makes use of the above CSS.
		www.w3schools.com
		www.codeacademy.com
		html.net/tutorials
		www.khanacademy.org
		
		Clydeview Academy materials
		
		Dalziel High School HTML teaching videos
		
		Teaching web design and implementation workshop

Implementation (HTML)	Describe, exemplify and implement HTML code: ♦ HTML ♦ head ♦ title ♦ body ♦ heading ♦ paragraph ♦ DIV ♦ link ♦ anchor ♦ IMG ♦ audio ♦ video ♦ lists — ol, ul and li Read and explain code that makes use of the above HTML.	www.w3schools.com www.codeacademy.com html.net/tutorials www.khanacademy.org Clydeview Academy materials Dalziel High School CSS teaching videos Teaching web design and implementation workshop

Appendix 1: design techniques (SDD)

Structure diagrams

A structure diagram is a method of graphically representing the steps required to solve a problem. Structure diagrams must be read from the top down from left to right.

There are four types of notations used to represent the workings of the program:

[rectangle]	Process	Notes a process such as a calculation.
[rounded rectangle]	Loop	Problem may repeat a mixed number of times or repeat if conditions are met.
[hexagon]	Selection	Problem may branch depending on the conditions met.
[predefined process box]	Predefined process	Shows use of a predefined function or a procedure.

Structure diagram: example 1
This diagram calculates the average of five numbers.
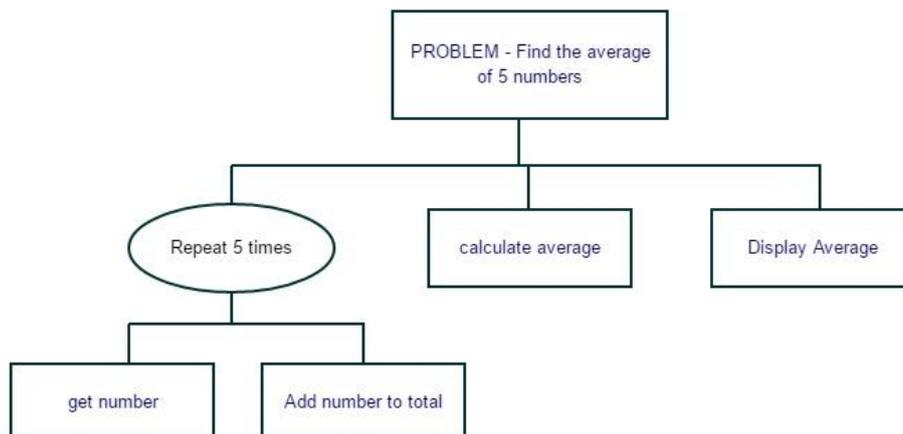
Structure diagram: example 2

This diagram decides the cost of the postage of a parcel.
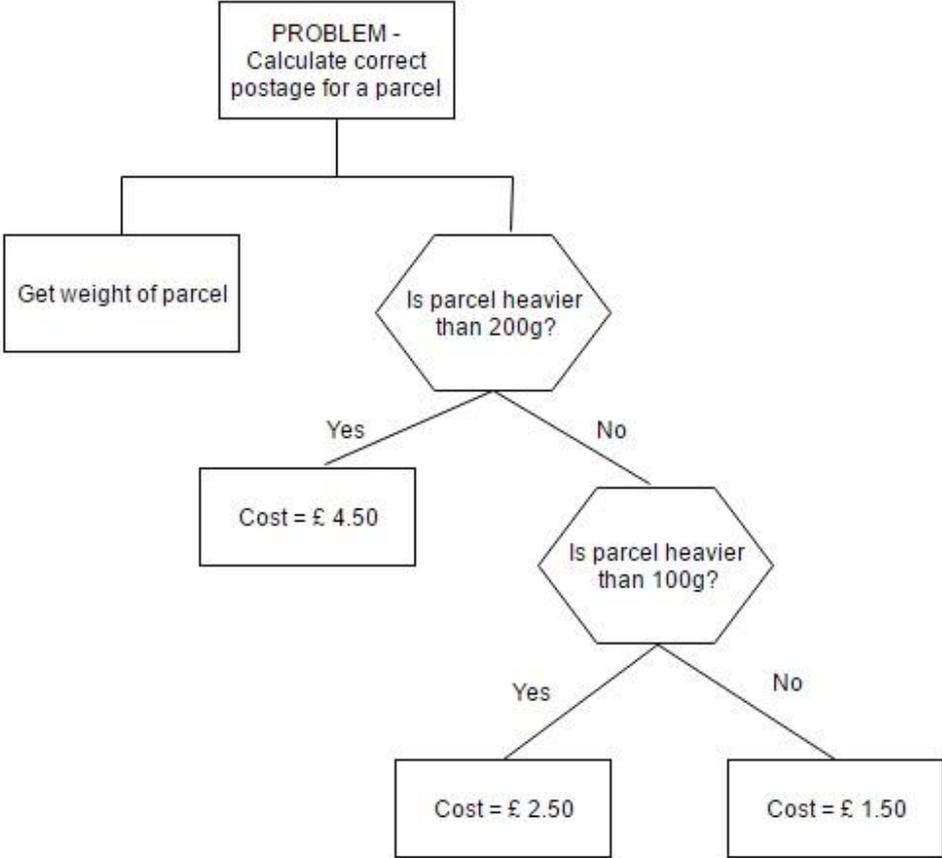
Flowcharts

A flowchart uses a variety of standard symbols with text to represent the order of events required to solve a problem. The symbols in a flowchart can be equated to programming constructs such as assignment, selection and repetition.

Flowchart symbols

Symbol	Name	Description
→	Flow line	Shows the direction or flow between symbols.
(rounded rectangle)	Terminal	Represents the "start" and "end" of a problem.
(hexagon)	Initialisation	Used to show declaration of variables/arrays or assignment of an initial value.
(parallelogram)	Input/output	Shows data input or output.
(diamond)	Decision	Problem may branch or repeat if conditions are met.
(rectangle)	Process	Notes a process such as a calculation.
(double-barred rectangle)	Predefined process	Shows use of a predefined function often with parameters.
(circle)	On-page connector	May be used to split a flowchart to keep it on a single page.
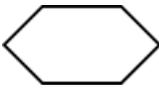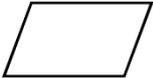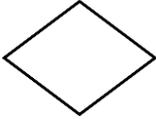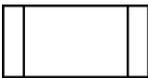
Flowchart: example 1

This flowchart inputs, adds and displays a value.
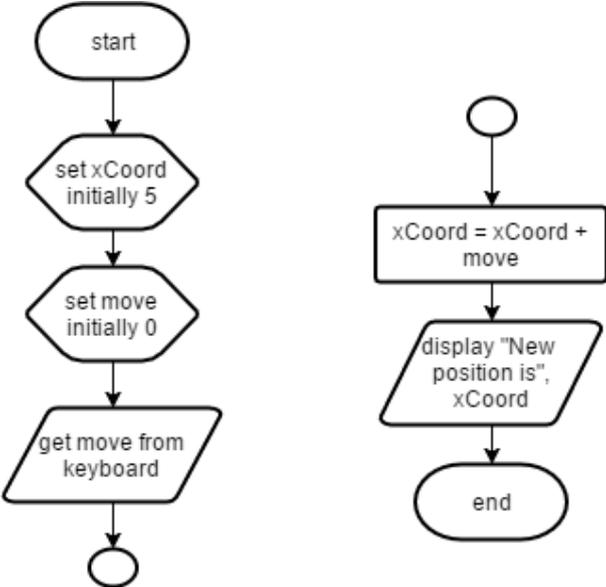
Flowchart: example 2
This flowchart checks to see if five different people are old enough to rent a van.
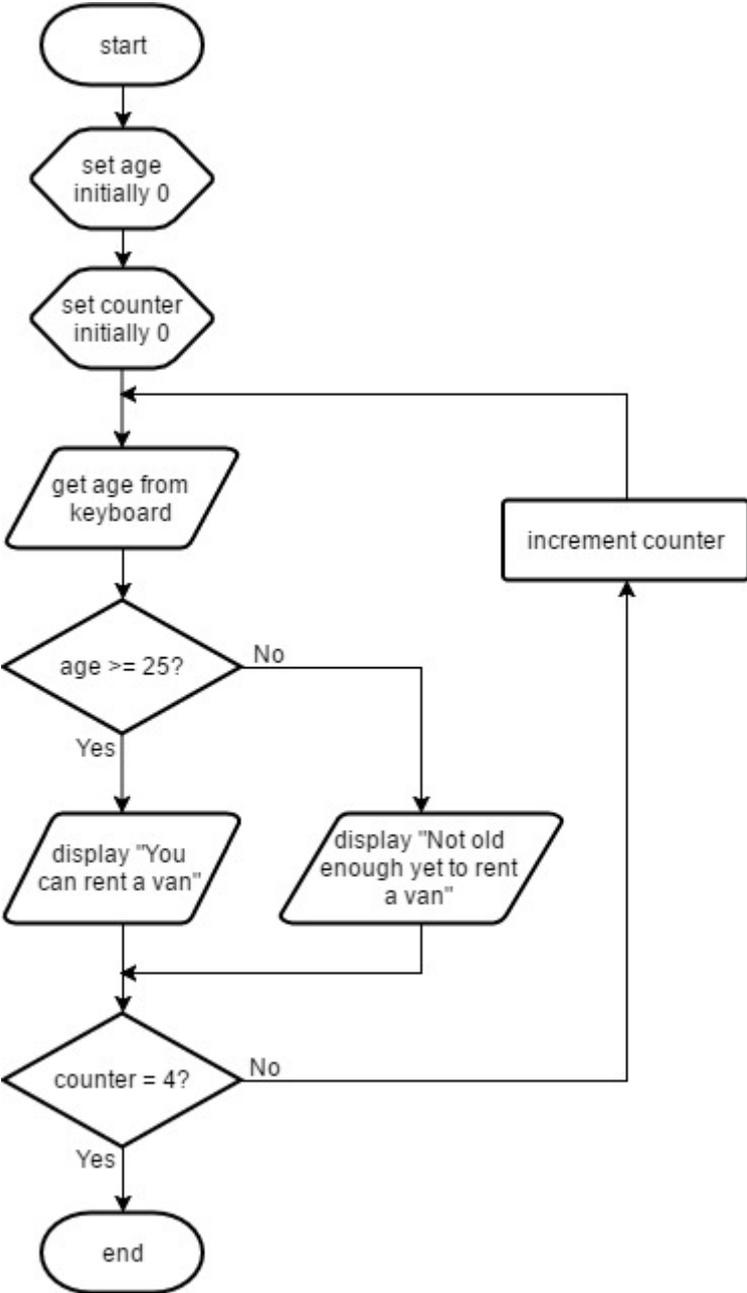
Flowchart: example 3
This flowchart keeps a running total of scores, until there are no more scores.
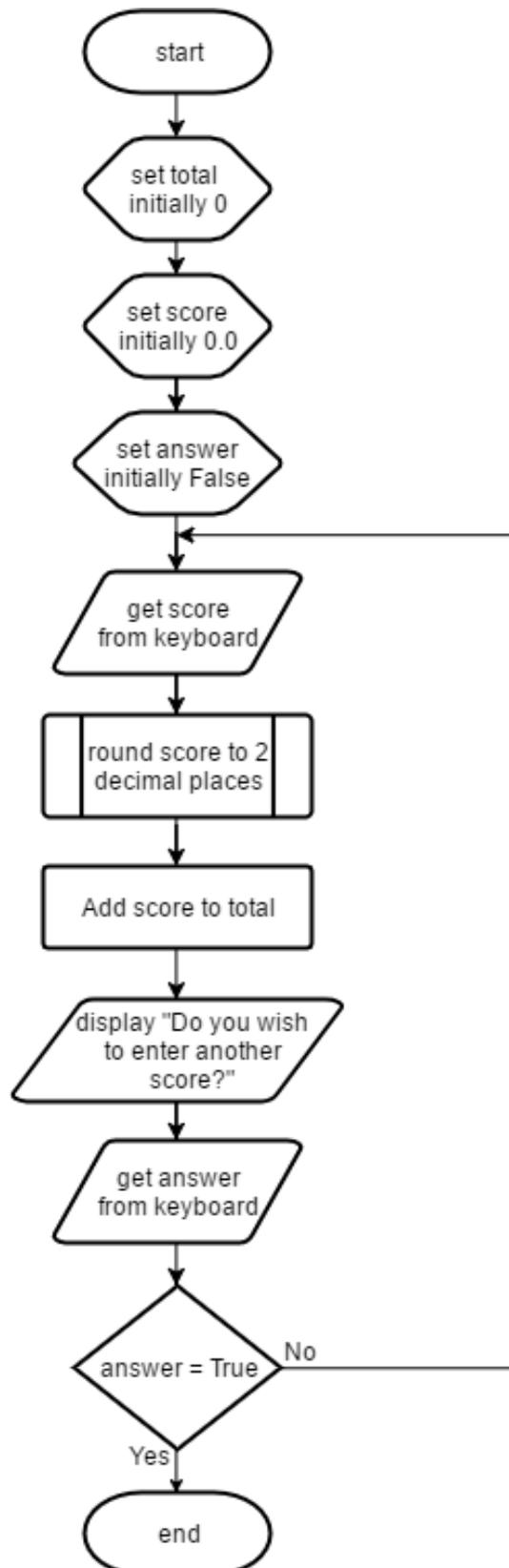
Pseudocode

Pseudocode is a natural language-based design methodology used to define an algorithm and refinement. It is a kind of structured English for describing algorithms and is intended for human reading. It typically omits details such as variable declarations and system-specific code. Pseudocode can look like code in an implemented problem but it doesn't have the same strict syntax.

The design should begin by defining the main steps or algorithm. Where a step of the algorithm requires refinement, a numbering system may be used to denote which line of the algorithm is being refined.

Identified selection and repetition in the problem, may be highlighted with indentation.

Pseudocode: example 1
This algorithm calculates the volume of a swimming pool.

Algorithm

```
1    Ask user to enter dimensions of a swimming pool in metres
2    Calculate volume of pool
3    Display message stating the volume of the pool
```

Refinement

```
1.1  Ask user to enter length of pool
1.2  Ask user to enter width of pool
1.3  Ask user to enter depth of pool

2.1  Volume is calculated as length * width * depth

3.1  Display "The volume of the pool is", volume
```

Pseudocode: example 2

This algorithm adds up the length of the tracks on a CD.

Algorithm

```
1 Initialise total length
2 Get valid number of tracks from user
3 Start fixed loop for each track
4   Get title and track length from user
5   Add track length to total
6 End fixed loop
7 Display track titles and track lengths
8 Display total length
```

Refinement

```
2.1  Start conditional loop
2.2     Get number of tracks from user
2.3     If number of tracks is not valid display error message
2.4  Repeat until the number of tracks entered is between 1 and 20
     inclusive

4.1  Get track title and store in names array
4.2  Get track length and store in length array

5.1  Add track length to total length

7.1  Start fixed loop for length of names array
7.2     Display "The name of track", counter, "is", track name
7.3     Display "The length of track", counter, "is", track length
7.4  End fixed loop

8.1  Display "The total length of the tracks is", total length
```

Appendix 2: user-interface design (SDD)

The user interface is the part of a computer program that is visible to the user. It can be as simple as a command line or as sophisticated as a virtual reality simulator. The point of designing a user interface for software is to show what input and output is required, so that the programmer can implement it in their chosen code.

The type of user interface can depend on what the programming language is capable of achieving. The examples below would probably be sketched by the designer, rather than typed.

User-interface design: example 1
This is a user-interface design for a program that calculates a user's weekly pay, using a text based language such as True Basic.

Prompt (computer)   **Response (user)**

Enter how many hours you worked this week   ___

Enter how many hours you worked on Saturday   ___

Enter how many hours you worked on Sunday   ___

Your pay this week is _____

User-interface design: example 2
This is a user-interface design for a program that calculates a user's weekly pay using an event-driven language such as Visual Basic.

Enter how many hours you worked this week

Enter how many hours you worked on Saturday

Click here to run

Enter how many hours you worked on Sunday

Your pay this week is

Appendix 3: standard algorithms (SDD)

There are three standard algorithms in the National 5 course specification. These are:

♦ input validation — checking that input is acceptable
♦ running total within a loop — adding up a list of values
♦ traversing a 1D array — accessing each element of an array from first to last

Each of these algorithms are exemplified below using SQA's 'Reference Language for Computing Science question papers'.

Input validation: example 1 (while loop)
This program is used to obtain a value between 10 and 20 inclusive.

```
RECEIVE number FROM KEYBOARD
WHILE number < 10 OR number > 20 DO
    SEND "Error, please enter again" TO DISPLAY
    RECEIVE number FROM KEYBOARD
END WHILE
```

Input validation: example 2 (until loop)
This program is used to obtain a value between 10 and 20 inclusive.

```
REPEAT
    RECEIVE number FROM KEYBOARD
    IF number < 10 OR number > 20 THEN
        SEND "Error, please enter again" TO DISPLAY
    END IF
LOOP UNTIL number >= 10 AND number <= 20
```

Running total within a loop: example 1 (fixed loop)
This program is used to calculate the sum of a known number of values entered by the user one at a time.

```
DECLARE total INITIALLY 0
FOR loop FROM 1 TO 10 DO
    RECEIVE number FROM KEYBOARD
    SET total TO total + number
END FOR
```

Running total within a loop: example 2 (conditional loop)
This program is used to calculate the sum of an unknown number of values entered by the user one at a time.

```
DECLARE total INITIALLY 0
REPEAT
     RECEIVE number FROM KEYBOARD
     SET total TO total + number
     SEND "Do you wish to enter another value" TO DISPLAY
     RECEIVE choice FROM KEYBOARD
LOOP UNTIL choice = "no"
```

Traversing a 1D array: example 1 (fixed loop)
This program is using a loop to access each element of an array, for the purposes of processing the data in the array.

```
DECLARE allScores INITIALLY [ 12,34,23,54,32,67,26,23 ]
FOR counter FROM 0 TO 7 DO
    IF allScores[counter] >= 50 THEN
        SEND "Great Score" & allScores[counter] TO DISPLAY
END IF
END FOR
```

Traversing a 1D array: example 2 (fixed 'for each' loop with running total included)
This program is using a loop to access each element of an array, for the purposes of processing the data in the array.

```
DECLARE allScores INITIALLY [ 12,34,23,54,32,67,26,23 ]
DECLARE total INITIALLY 0
DECLARE counter INITIALLY 0
FOR EACH FROM allScores DO
    SET total TO total + allScores[counter]
SET counter TO counter + 1
END FOR
```

Appendix 4: efficient use of coding constructs (SDD)

A computer program can be written in many ways to solve a problem. Although each solution can achieve the same result by using the constructs available in a programming language, the programmer should be trying to achieve the most efficient solution.

Repetition

The use of repetition can greatly reduce the number of lines of code that have to be typed.

This program finds the average of 10 numbers.

```
DECLARE total INITIALLY 0
RECEIVE number FROM KEYBOARD
SET total TO total + number
RECEIVE number FROM KEYBOARD
SET total TO total + number
RECEIVE number FROM KEYBOARD
SET total TO total + number
RECEIVE number FROM KEYBOARD
SET total TO total + number
RECEIVE number FROM KEYBOARD
SET total TO total + number
RECEIVE number FROM KEYBOARD
SET total TO total + number
RECEIVE number FROM KEYBOARD
SET total TO total + number
RECEIVE number FROM KEYBOARD
SET total TO total + number
RECEIVE number FROM KEYBOARD
SET total TO total + number
RECEIVE number FROM KEYBOARD
SET total TO total + number
SET average TO total / 10
```

This program does exactly the same but uses more efficient constructs.

```
DECLARE total INITIALLY 0
REPEAT 10 TIMES
        RECEIVE number FROM KEYBOARD
        SET total TO total + number
END REPEAT
SET average = total / 10
```

1D arrays

1D arrays allows the same variable name to be used to store a list of similar variables values. This means that repetition can be used to easily store values that may be required later.

This program finds the average of 10 numbers and stores each of the numbers that were input.

```
DECLARE total INITIALLY 0
RECEIVE number1 FROM KEYBOARD
RECEIVE number2 FROM KEYBOARD
RECEIVE number3 FROM KEYBOARD
RECEIVE number4 FROM KEYBOARD
RECEIVE number5 FROM KEYBOARD
RECEIVE number6 FROM KEYBOARD
RECEIVE number7 FROM KEYBOARD
RECEIVE number8 FROM KEYBOARD
RECEIVE number9 FROM KEYBOARD
RECEIVE number10 FROM KEYBOARD
SET total TO number1 + number2 + number3 + number4 + number5 +
number 6 + number7 + number8 +number9 + number10
SET average TO total / 10
SEND average TO display
<display the ten values>
```

This program does exactly the same, but uses more efficient constructs and data structures.

```
DECLARE number INITIALLY []
FOR counter FROM 1 TO 10 DO
    RECEIVE number[counter)]
    SET total TO total + number[counter]
END FOR
SET average TO total/10
SEND average TO display
```

Selection

Choosing from a number of possible alternatives when using selection can make code more efficient, however, it is not always obvious which is more efficient.

These two programs decide the grade that a candidate is given, depending on the mark they received in the exam.

Example 1		**Example 2**
This uses four IF constructs, one after another, with the use of complex conditional statements.		This uses nested IF constructs with simple conditional statements. Other programming languages may use a CASE, ELIF or ELSEIF statement.

<pre>IF mark < 50 THEN
     SET grade TO D
END IF
IF mark>=50 AND mark<=59 THEN
    SET grade TO C
END IF
IF mark>=60 AND mark<=69 THEN
    SET grade TO B
END IF
IF mark>=70 THEN
    SET grade TO A
END IF</pre>
		<pre>IF mark>=70 THEN
		SET grade=A
		ELSE
		    IF mark>=60 THEN
		        SET grade=B
		    ELSE
		        IF mark>=50 THEN
		            SET grade=C
		         ELSE
		            SET grade=D
		        END IF
		    END IF
		END IF</pre>

This program always carries out four comparisons, regardless of the values stored in mark.		This program carries out either one, two or three comparisons, depending on the values stored in the mark.

Logical operators

Logical operators can be useful when creating complex conditions, rather than using multiple simple conditions.

This program uses two simple conditional statements.

```
IF X > 4 THEN
   IF Y < 6 THEN
    SET quadrant TO 2
   END IF
END IF
```

This program uses one complex conditional statement.

```
IF X > 4 AND Y < 6 THEN
   SET quadrant TO 2
END IF
```

Appendix 5: design: entity-relationship diagrams (DDD)

An entity-relationship diagram is a graphical representation of the entities in a system. It is used to illustrate the relationship that exists between two or more entities.

Although several different representations can be used, the entity-relationship diagrams shown in the following examples use crow's feet notation to indicate the many sides of the relationship.

A relational database is used by a travel agency to store details of Scottish holiday resorts and hotels in each resort. The resort and hotel details have been arranged in two entities.

Entity: Resort
<u>resortID</u>
town
resortType
trainStation

Entity: Hotel
<u>hotelRef</u>
hotelName
phoneNumber
resortID*
starRating
seasonStartDate
swimmingPool
mealPlan
checkInTime

Note: primary keys are underlined. Foreign keys are marked with an asterisk.

Example 1: entities and attributes
This diagram illustrates the entities together with their attributes

Example 2: entities
This diagram shows only the entities.

Appendix 6: design — data dictionary (DDD)

A data dictionary is used to indicate the properties of each attribute needed to define the entities.

Example
A relational database is used by a travel agency to store details of Scottish holiday resorts and hotels. The resort and hotel details are arranged in two separate entities.

This shows sample data for each table of the database:

Resort table

Resort ID	Town	Resort Type	Train Station
168	Ayr	coastal	true
347	Portree	island	false

Hotel table

Hotel Ref	Hotel Name	Phone Number	Resort ID	Star Rating	Season Start Date	Swimming Pool	Meal Plan	Check-In Time
AY72	Cliff Top	01292123456	168	3	29/04/2017	false	HB	14:30
PR04	Bay View	01478456789	347	5	01/05/2017	true	BB	16:00
AY19	Glee	01292987654	168	2		false	FB	15:00

This shows the completed data dictionary:

Entity: Resort					
Attribute Name	**Key**	**Type**	**Size**	**Required**	**Validation**
resortID	PK	Number		yes	
town		Text	20	yes	
resortType		Text	20	yes	Restricted choice: coastal, city, island
trainStation		Boolean		yes	
Entity: Hotel					
Attribute Name	**Key**	**Type**	**Size**	**Required**	**Validation**
hotelRef	PK	Text	4	yes	length=4
hotelName		Text	20	yes	
phoneNumber		Text	11	yes	length=11
resortID	FK	Number		yes	Existing resortID from Resort table
starRating		Number		yes	Range: >=1 and <=5
seasonStartDate		Date		no	
swimmingPool		Boolean		yes	
mealPlan		Text	17	yes	Restricted choice: RO, BB, HB, FB
checkInTime		Time		yes	Range: >=14:00 and <=16:00

Appendix 7: design of solution to database queries (DDD)

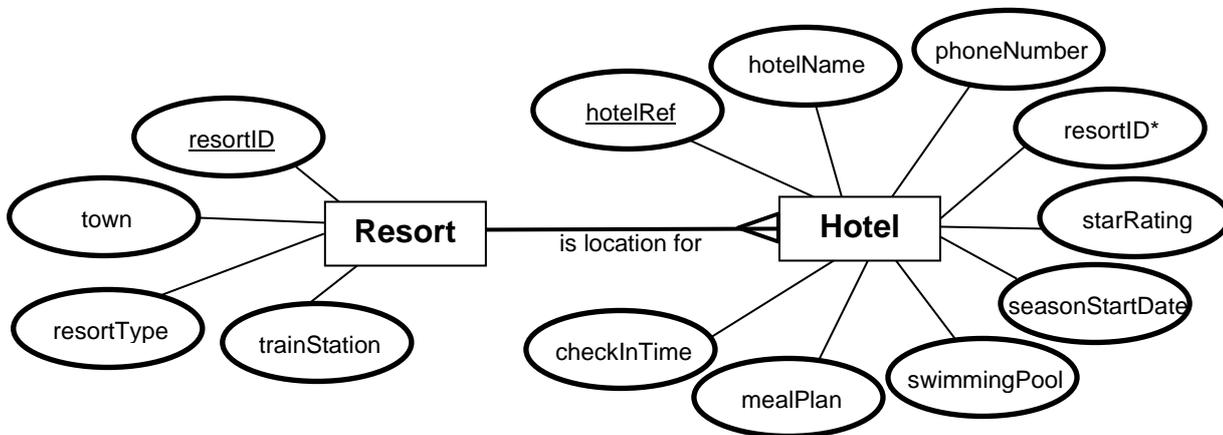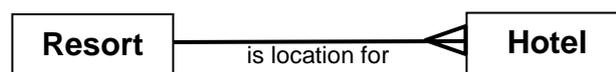A relational database is used by a travel agency to store details of Scottish holiday resorts and hotels in each resort. The resort and hotel details are arranged in two separate tables called **Resort** and **Hotel**. The structure of the tables is shown below:

Resort

	Field Name
🔑	resortID
	town
	resortType
	trainStation

Hotel

	Field Name
🔑	hotelRef
	hotelName
	phoneNumber
	resortID
	starRating
	seasonStartDate
	swimmingPool
	mealPlan
	checkInTime

Planning the design of a query before creating the SQL code is good practice. This gives candidates time to think carefully about the fields that are required, which in turns, helps them to identify the table or tables that will be needed. It also allows candidates to consider the purpose of the query (search and/or sort), together with any required search criteria and/or sort order. Planning ahead helps to reduce the errors that candidates may otherwise encounter when working with the SQL code.

A simple table template, such as the one shown below, can be used by candidates to indicate the planned design of a SQL query.

Example 1
Design a query to list the town name and train station details of all resorts that have a train station.

Field(s)	town, trainStation
Table(s)	Resort
Search criteria	trainStation = true
Sort order	

Example 2
Design a query to list the hotel name and phone number, together with the star rating and swimming pool details for all hotels with a swimming pool that have a rating of at least 4 stars.

Field(s)	hotelName, phoneNumber, starRating, swimmingPool
Table(s)	Hotel
Search criteria	swimmingPool = true AND starRating >= 4
Sort order	

Example 3
Design a query to list the hotel name and phone number, together with the town and train station details of any hotel in Ayr.

Field(s)	hotelName, phoneNumber, town, trainStation
Table(s)	Hotel, Resort
Search criteria	town = "Ayr"
Sort order	

Example 4
Design a query to list the town name, resort type and star rating of all hotels that have a 5 star rating. These details should be listed in alphabetical order of town.

Field(s)	town, resortType, starRating
Table(s)	Resort, Hotel
Search criteria	starRating = 5
Sort order	town ASC

Example 5
Design a query to list the hotel name and its star rating, together with the town, resort type and check-in time, of all hotels that allow check in before 15:00. These details should be displayed so that the hotel with the highest rating is listed first; hotels with the same star rating should be listed in alphabetical order of town name.

Field(s)	hotelName, starRating, town, resortType, checkInTime
Table(s)	Hotel, Resort
Search criteria	checkInTime < 15:00
Sort order	starRating DESC, town ASC

Appendix 8: testing and evaluation (DDD)

Consider the **Customer** table shown below.

custo	foreName	surname	street	town	package	directDebit	paymentDu
101	Lauren	Calder	2 Paisley Street	Wemyss Bay	Premier	✔	12/04/2016
102	Abigail	Cameron	16 Paisley Street	Wemyss Bay	Standard	☐	12/04/2016
103	Ryan	Collins	17 Dunoon Drive	Gourock	Premier	✔	19/05/2016
104	Nicole	Rutherford	5A Panama Place	Port Glasgow	Large	☐	13/04/2016
105	Justine	O'Docherty	7 High Street	Kilmacolm	Premier	✔	18/04/2016
106	Shelby	Sweeney	3 Paisley Road	Port Glasgow	Premier	✔	26/05/2016
107	Donald	McAndrew	1 Big Hill Avenue	Inverkip	Standard	✔	01/08/2016
108	Rowan	Hastings	6 Clydeview Crescent	Gourock	Large	✔	05/05/2016
109	Grant	Donaldson	9 Dunoon Drive	Gourock	Premier	☐	07/04/2016
110	Christine	Flowers	63 Hamilton Drive	Greenock	Large	✔	19/04/2016
111	Ross	Lambie	12 Paisley Road	Kilmacolm	Standard	✔	27/03/2016
112	Paul	McGill	3C Cow Lane	Kilmacolm	Premier	☐	10/04/2016
113	Jack	Shields	4 Brookside Close	Port Glasgow	Large	☐	10/05/2016
114	Pauline	Milne	32 High Street	Kilmacolm	Premier	✔	09/05/2016
115	Margaret	Rice	5 Drumchapel Square	Greenock	Standard	☐	14/04/2016

Query testing

A query is required to display the full name and town of all customers who live in Gourock.
The details should be listed in alphabetical order of customer surname.

This table shows the output predicted from the query.

Expected output	Forename	Surname	Town
Details of customer listed **first**	Ryan	Collins	Gourock
Details of customer listed **last**	Rowan	Hastings	Gourock

Query evaluation

This answer table is the output from the query used to perform the task.

surname	town
Collins	Gourock
Donaldson	Gourock
Hastings	Gourock

Actual output
Comparing the answer table with the predicted output, it is possible to evaluate the query.

The query is fit for purpose because it displays details of the three customers who live in Gourock and has arranged the details in ascending order of surname. However, the query output is not accurate because the answer table only shows details of surname and town; the forenames of the customers are missing from the answer table.

Appendix 9: website structure (WDD)

The design of a website should indicate the type of **navigational structure** that will be used — usually linear or hierarchical. The design should also show the direction of each of the links.

Example

A new website for ScotsWaterSport is being developed. The website will consist of five web pages and each of these web pages will have a main heading centred at the top of the page. Further requirements for the web pages are as follows.

The home page will provide:

♦ a short introduction to the range of water sports available in Scotland
♦ internal hyperlinks to specialist pages about four different water sports (Kayaking, Surfing, Rafting, and Sailing)
♦ one external link to the water sports page of the VisitScotland website

Each of the specialist sports pages will provide:

♦ a photo of the sport
♦ a paragraph of information about the sport
♦ a bulleted list of suggested locations and ideas to try out the sport
♦ a hyperlink back to the home page

The new website for ScotsWaterSport will have a hierarchical structure.

This diagram shows the navigational structure of the ScotsWaterSport website. The arrows on the diagram indicate the direction of the hyperlinks provided on each page.
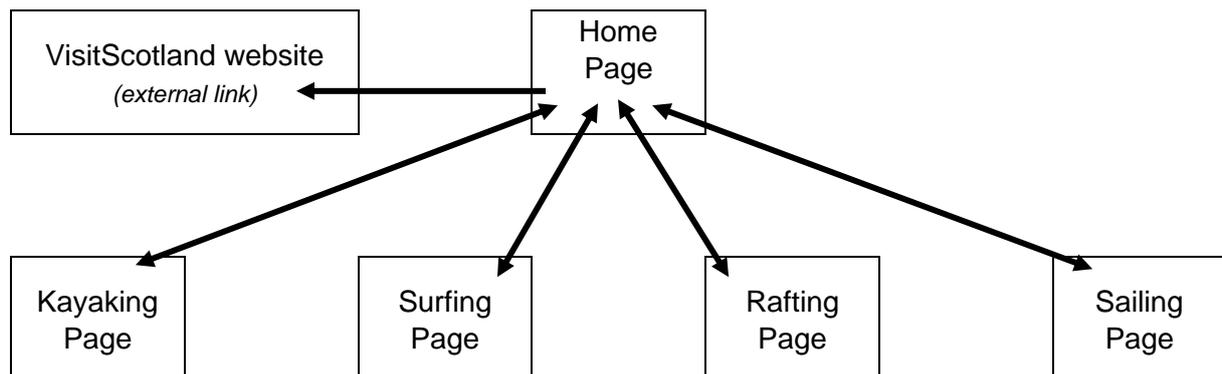
Appendix 10: interface design (WDD)

The user-interface planning can be illustrated using **wireframes**. A separate wireframe is needed for each page on a website. Each wireframe indicates the intended layout of the page and shows the position of:

♦ all text elements on the page

♦ any media elements (images, audio clips and video clips)

♦ elements that allow the user to interact with the page

♦ intended position and type of all hyperlinks on the page

For the National 5 course, content should be stacked vertically on the page. It is not expected that media will be positioned side by side

Example

A new website for ScotsWaterSport is being developed. The website will consist of five web pages and each of these web pages will have a main heading centred at the top of the page. Further requirements for the web pages are provided below.

The home page will provide:

♦ a short introduction to the range of water sports available in Scotland

♦ internal hyperlinks to specialist pages about four different water sports (Kayaking, Surfing, Rafting and Sailing)

♦ one external link to the water sports page of the Visit Scotland website

Each of the specialist sports pages will provide:

♦ a paragraph of information about the sport

♦ a photo of the sport

♦ a bulleted list of suggested locations to try out the sport

♦ a hyperlink back to the home page

These wireframes show the planned layout of each page on the ScotsWaterSports website.

ScotsWaterSports - Home Page

ScotsWaterSports HomePage

Intro parragraph

Internal hyperlink to Kayaking Page

Internal hyperlink to Surfing Page

Internal hyperlink to Rafting Page

Internal hyperlink to Sailing Page

External hyperlink to Visit Scotland Page about water sports

ScotsWaterSports - Kayaking Page

Kayaking Page

Information about kayaking in Scotland

kayaking.jpg

Intro to list of suggestions
- suggestion 1
- suggestion 2
- suggestion 3

Internal hyperlink to Home Page

Surfing Page

Information about surfing in Scotland

surfing.jpg

Intro to list of suggestions
- suggestion 1
- suggestion 2
- suggestion 3

Internal hyperlink to Home Page

Rafting Page

Information about rafting in Scotland

rafting.jpg

Intro to list of suggestions
- suggestion 1
- suggestion 2
- suggestion 3

Internal hyperlink to Home Page

Appendix 11: low-fidelity prototyping (WDD)

A prototype is used to show the intended user interface for any software product. Once developed, a prototype forms a critical component of usability testing, along with personas (or detailed descriptions of typical end users) and test cases. Prototyping can be low-fidelity or high-fidelity, with developers sometimes using both.

Low-fidelity prototypes are paper-based. They can be created quickly and give potential end users of the finished product an indication of how the product will look and feel as they interact with it.

Although not in the National 5 course, for clarification purposes, High-fidelity prototypes are electronic. They are often created using RAD tools, meaning that generating a working interface doesn't take long as, at this stage, only the interface is built. The detail of any processes needed is ignored and there is no functionality behind any of the screen widgets (other than to move screen or display messages such as 'new content is displayed here'). A high-fidelity prototype lets potential end users 'play' with the interface as it will be in the finished product. As a result, it gives a much more realistic idea of how the finished product will look and feel as it is being used, than would be possible with a low-fidelity, paper-based prototype.

During usability testing, a selection of end users (or testers who adopt the personas) are asked to complete the task described in each of the test cases using the prototype. Developers will be on hand to 'change screen' (show a new page of the interface) or 'update the content of the screen' (for example, replace page content, show the results of a calculation or perform an interactivity) as the user interacts with the widgets used on the prototype. By listening to user feedback and observing any difficulties users have as they perform the specified tasks, the developers can make changes and improvements to the user interface at an early stage in the development of the software product.

Prototypes should be based on the layout indicated in the wireframes. However, unlike wireframes which are created to ensure consistency and share details with members of the development team, the intended audience of prototypes is end users of the finished product. For this reason, a prototype should show more details for the content and the screen widgets that will be used to perform tasks.

Creating low-fidelity prototypes

Low-fidelity prototypes can be created in a number of ways. For example:

♦ A simple hand-drawn sketch of the proposed interface is one of the easiest and cheapest ways of creating a paper-based prototype. Coloured pencils, felt pens and markers can be used to add colour; quick hand drawings of images and widget icons will give end users a good idea of what is intended. Any size of paper can be used but flip-chart paper can be easier for candidates to handle and facilitates collaborative group work.

♦ 'Pencil' software by 'Evolus' provides free prototyping tools. A number of in-built templates are provided and additional Android and iOS templates are available for download. 'Pencil' templates can be used to create realistic-looking interfaces that can be exported as PNG files and printed to generate prototypes for usability testing. See http://pencil.evolus.vn/

♦ Graphics packages such as 'Paint' can be used to create the intended layout using tools provided in the toolbox. See https://www.getpaint.net/

Example
A new website for ScotsWaterSport is being developed. The wireframes for each of the pages have been created and prototyping of the Home Page and Kayaking Page of the ScotsWaterSports website can now take place.

Version 1
These prototypes were created with marker pens and flip-chart paper.

Here are a few suggestions for kayaking in Scotland.

- River Tay - one of Scotland's longest and most famous rivers offers steady conditions for beginners and experienced paddlers
- Loch Lomond - paddle on the largest loch of fresh water in Great Britain
- Isle of Skye - visit the sea cliffs, arc and tunnels all along the spectacula coastline

Return to Home Page.

Version 2
These prototypes were created using 'Pencil' templates.

ScotsWaterSports – Home Page   ☐ ◻ ✕

ScotsWaterSports Home Page

A wide range of water sports are available in Scotland. Exploring the scenery and wilderness of Scotland by paddle offers something for everyone. Find out more by checking out the links below.

Kayaking

Surfing

Rafting

Sailing

For more information about water sports in Scotland, see Water Sports in Scotland

ScotsWaterSports – Kayaking Page   ☐ ◻ ✕

Kayaking Page

Many different types of kayaking are available at locations across Scotland. Paddle through some of the country's most beautiful scenery on a Scottish loch or river. Or explore caves and secluded beaches by taking a sea kayaking trip along our stunning coastline.

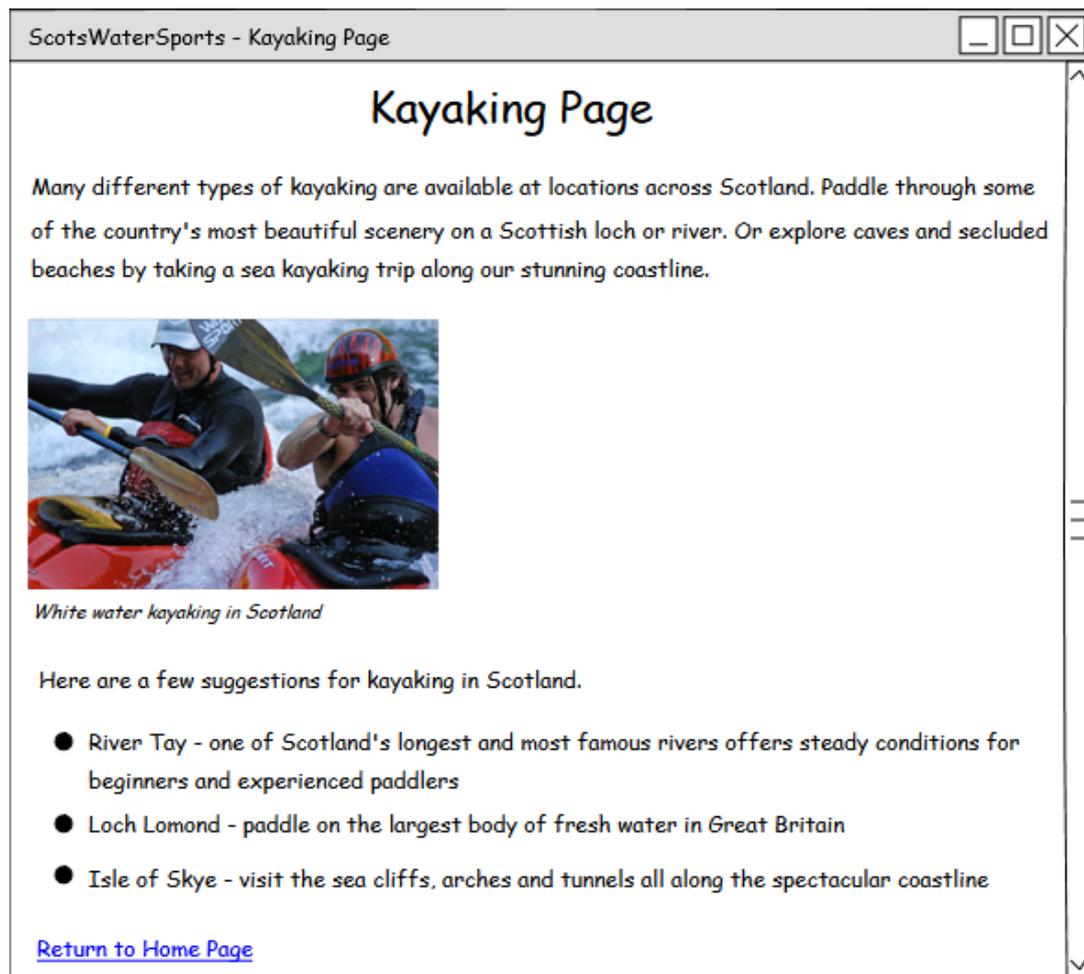*White water kayaking in Scotland*

Here are a few suggestions for kayaking in Scotland.

● River Tay - one of Scotland's longest and most famous rivers offers steady conditions for beginners and experienced paddlers
● Loch Lomond - paddle on the largest body of fresh water in Great Britain
● Isle of Skye - visit the sea cliffs, arches and tunnels all along the spectacular coastline

Return to Home Page

Appendix 12: SQL (DDD)

SQL stands for 'Structured Query Language'. SQL is a special purpose programming language for storing, manipulating and retrieving data in relational databases. Although most database systems use SQL, there can be a number of differences between different dialects or versions. However, the standard SQL commands such as SELECT, INSERT, UPDATE and DELETE are common to them all.

We will use the following relational database to exemplify the SQL commands:

The Scottish Handball League use a relational database to store details of teams and players in two separate tables called `Team` and `Player`. The structure of these tables is shown below:

Player
Field Name
🔑 playerID
firstname
surname
dateOfBirth
position
goalsScored
teamName

Team
Field Name
🔑 teamName
town
leagueDivision
district
trainingVenue
contactPerson
emailAddress

A sample record stored in each table is shown below:

Player table	
playerID	810JE
firstname	Jack
surname	Edwards
dateOfBirth	03/05/1994
position	Left back
goalsScored	37
teamName	Clyde Flyers

Team table	
teamName	Clyde Flyers
town	Greenock
leagueDivision	1
district	West
trainingVenue	Burnside Sports Centre
contactPerson	Chris Black
emailPerson	clydeflyers@handball.mail.uk

Searching

The **SELECT** statement is used to decide which fields should be displayed. The statement is followed by the fields, separated by commas.

The **FROM** clause states the names of the database table(s) that are needed in the query.

The **WHERE** clause states the criteria that must be met. This clause is followed by the field name, an operator (<, >, =) and the information inside inverted commas if it is text.

> **SELECT** *fieldName1, fieldName2, fieldName3, etc*
> **FROM** *tableName*
> **WHERE** *fieldName = data;*

Example 1 To search the database to display the town, contact person, e-mail address and district for all teams in the west district, you would write the following SQL.

```
SELECT town, contactPerson, emailAddress, district
FROM Team
WHERE district = "West";
```

Logical operators (AND, OR) can be used to create complex criteria.

Example 2 To search the database to display the surname, position, and division of all players who either play in the right wing or centre position, you would write the following SQL.

```
SELECT surname, position, division
FROM Player
WHERE position = "right wing"
OR position = "centre";
```

EQUI-JOIN between tables

If a search involves displaying data found in two linked tables the SQL clause must state the link. An **EQUI-JOIN** is added to the **WHERE** statement stating that the primary and foreign key values in both tables must match.

> **SELECT** *fieldName1, fieldName2, fieldName3*
> **FROM** *tableName1, tableName2*
> **WHERE** *tableName1.fieldNamePK = tableName2.fieldNameFK*
> ***AND** fieldName = data;*

Example 3 To search the database to display the full name and town of any players who play for a handball team based in Paisley, you would write the following SQL.

```
SELECT firstname, surname, town
FROM Team, Player
WHERE Team.teamName = Player.teamName
AND town = "Paisley";
```

Example 4   To search the database to display the full name, number of goals scored and the team name of any players who have scored at least 20 goals for Lothian Flames, you would write the following SQL.

As the teamName field appears in both tables, you have to specify which table you want to use, eg Team.teamName

```
SELECT firstname, surname, goalsScored, Team.teamName
FROM Team, Player
WHERE Team.teamName = Player.teamName
AND goalsScored >= 20
AND Team.teamName = "Lothian Flames";
```

The following SQL would give the same solution for the query, but this time displaying the Player.teamName field.

```
SELECT firstname, surname, goalsScored, Player.teamName
FROM Team, Player
WHERE Team.teamName = Player.teamName
AND goalsScored >= 20
AND Team.teamName = "Lothian Flames";
```

Sorting

The **ORDER BY** clause decides how the output of the search should be sorted. ORDER BY is followed by the name of the field and then whether it is ascending (ASC) or descending (DESC) order.

> **SELECT** *fieldName1, fieldName2, fieldName3*
> **FROM** *tableName*
> **WHERE** *fieldName = data*
> **ORDER BY** *fieldName ascending or descending;*

Example 5   To search the database to display the full name, team name and goals scored for any player who has scored fewer than 30 goals this season, so that the player with the most goals is listed first, you would write the following SQL.

```
SELECT firstname, surname, Player.teamName, goalsScored
FROM Player
WHERE goalScored < 30
ORDER BY goalScored DESC;
```

Example 6   To search the database to display the full name, position, and team name of all goalkeepers who play in the league (listing in alphabetical order of surname; players with the same surname should be listed in alphabetical order of first name), you would write the following SQL.

```
SELECT firstname, surname, position, Player.teamName
FROM Player
WHERE position = "goalkeeper"
ORDER BY surname ASC, firstname ASC;
```

Adding records

The **INSERT INTO** statement is used to add records to a table. The statement is followed by the table name and then the **VALUES** statement, followed by the data in brackets separated by commas.

> **INSERT INTO** *tableName (fieldName1, fieldName2)*
> **VALUES** *(value1, value2);*

You must ensure that the order of the values is the same as the order of the fields.

Example 7   Details of the newest team to join the league are shown below.

Team name	Dundee Dynamos
Town	Dundee
Contact person	Paul McLaughlin
E-mail address	dynamos@dundeehandball.gmail.com
Division	Second
District	North
Training venue	DISC

When adding a complete record to a database, you could write the following SQL statement using only the values. There must be values for every field and they must be in the same order as the field names in the table.

```
INSERT INTO Team
VALUES ("Dundee Dynamos", "Dundee", "Paul McLaughlin",
"dynamos@dundeehandball.gamil.com", "Second", "North", "DISC");
```

When adding only partial record data to a table, both the field names and their associated values must be identified in the SQL statement.

Example 8   A new player has joined the team. The available details about the player are shown below.

Player ID	419AC
First name	Anatol
Surname	Czaja
Team name	Harris Hurricanes

To add these details to the database you would write the following SQL.

```
INSERT INTO Player (playerID, firstname, surname, teamName)
VALUES ("419AC", "Anatol", "Czaja", "Harris Hurricanes");
```

Editing records

The **UPDATE** statement is used to alter records in a table. The statement is followed by the name of the table, a **SET** clause and the **WHERE** clause which states what criteria must be met.

> **UPDATE** *tableName*
> **SET** *fieldName to updated value*
> **WHERE** *criteria to be met;*

Example 9   The player whose ID is 726HB has transferred to a new team and now plays for the Dundee Demons.

To update the correct record of the database you would write the following SQL.

```
UPDATE Player
SET Player.teamName = "Dundee Demons"
WHERE   playerID = "726HB";
```

Example 10   The contact details for the Airdrie Lions have changed. The team's contact person is now Lynne Jack and the team e-mail address is now airdrie@lionshandball.com.

To update the correct record of the database, you would write the following SQL.

```
UPDATE Team
SET contactPerson = "Lynne Jack", emailAddress =
"airdrie@lionshandball.com"
WHERE Team.teamName = "Airdrie Lions";
```

Deleting records

The **DELETE FROM** statement is used to delete records in a table. The statement is followed by the name of the table and the **WHERE** clause which states what criteria must be met.

> **DELETE FROM** *tableName*
> **WHERE** *criteria to be met;*

Example 11   The Borders Bandits have been knocked out of the league.

To remove the correct record from the database, you would write the following SQL.

```
DELETE FROM Team
WHERE Team.teamName = "Borders Bandits";
```

Example 12   Jack Roberts no longer plays in the handball league.

To remove the correct record from the database, you would write the following SQL.

```
DELETE FROM Player
WHERE firstname = "Jack"
AND surname = "Roberts";
```

Appendix 13: analysis (WDD)

During the analysis stage of website development, the following requirements should be identified:

1 End-user requirements:

♦ The end users are the people who are going to be using the website.

♦ Their requirements are the tasks they expect to be able to do using the website.

2 Functional requirements:

♦ Processes and activities that the system has to perform.

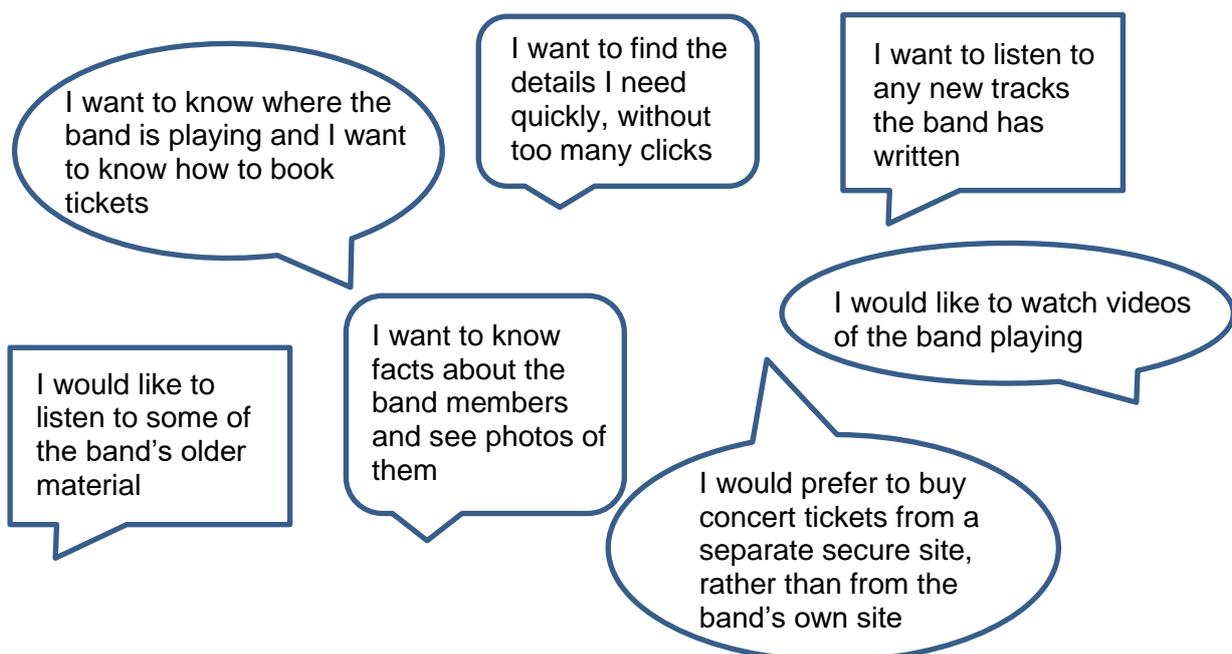♦ Information that the system has to contain to be able to carry out its functions.

These requirements will help:

♦ clarify the design of each webpage

♦ identify the features to be implemented on the website

♦ evaluate whether the system is fit for purpose after development is complete

Example

A rock band has three members. The band wants to develop a website for its fans. The site will provide details about the band, including biographies, music tracks, video clips and concert details.

The band asked some of its fans what they would like to see on the new website. Here are a few of the comments they made.

I want to know where the band is playing and I want to know how to book tickets

I want to find the details I need quickly, without too many clicks

I want to listen to any new tracks the band has written

I would like to watch videos of the band playing

I would like to listen to some of the band's older material

I want to know facts about the band members and see photos of them

I would prefer to buy concert tickets from a separate secure site, rather than from the band's own site

End-user requirements

☑ Users should be able to:

♦ navigate the site easily
♦ view biographies and photos of band members
♦ view all upcoming concerts and link to an external booking site
♦ view video clips of the band
♦ listen to the band's audio tracks

Functional requirements

☑ The **Home** page should provide internal links to the four topic pages (biographies, music, videos and concerts).
☑ **Individual profile** pages should include biography information, with photos of the band member and should have a link back to the **Biographies** page.
☑ The **Music** page should list the band's albums and allow individual audio tracks to be played.
☑ The **Videos** page should list video clips and allow these to be played.
☑ The **Concerts** page should list all the upcoming concerts, with links to the external booking site.
☑ All pages (except the **Home** page) should link back to the **Home** page.

Appendix 14: analysis (DDD)

During the analysis stage of database development, the following requirements should be identified:

1 End-user requirements:

- ♦ The end users are the people who are going to be using the database.
- ♦ Their requirements are the tasks they expect to be able to do using the database.

2 Functional requirements:

- ♦ Processes and activities that the system has to perform.
- ♦ Information that the system has to contain to be able to carry out its functions.
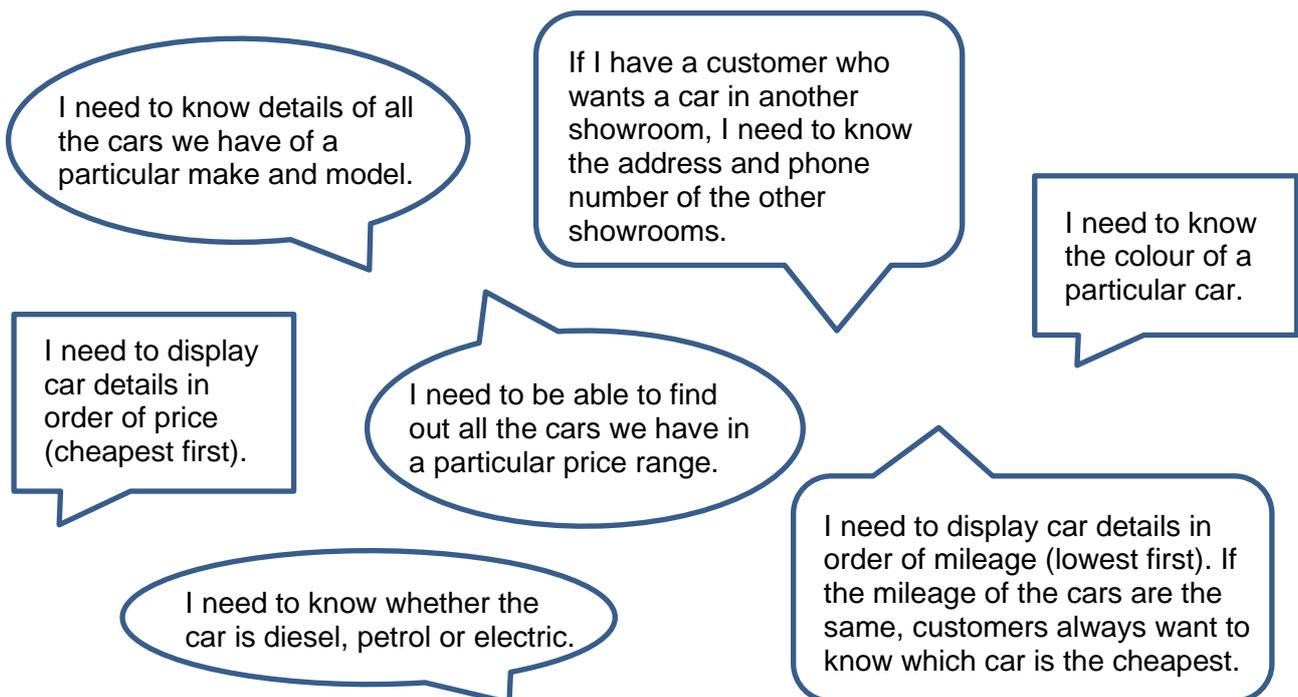
These requirements will help:

- ♦ clarify the design of the database
- ♦ identify the features to be implemented on the database
- ♦ evaluate whether the system is fit for purpose after development is complete

Example

A used-car dealership has six showrooms in different locations across Scotland. It wants to create a relational database to store details of cars owned by the company and details of each of their showrooms. The database will allow sales staff to view details of specific cars and to see which showroom the cars are located in.

The developers have asked some of the sales staff what they would like to see in the database. Here are a few of the comments they made.

I need to know details of all the cars we have of a particular make and model.

If I have a customer who wants a car in another showroom, I need to know the address and phone number of the other showrooms.

I need to know the colour of a particular car.

I need to display car details in order of price (cheapest first).

I need to be able to find out all the cars we have in a particular price range.

I need to display car details in order of mileage (lowest first). If the mileage of the cars are the same, customers always want to know which car is the cheapest.

I need to know whether the car is diesel, petrol or electric.

End-user requirements

☑ Sales staff should be able to display details of cars, by performing a number of different searches using:

♦ car make and model
♦ car colour
♦ range of car prices
♦ type of fuel used

☑ Search results should display:

♦ car make
♦ car model
♦ car price
♦ car fuel
♦ car mileage
♦ branch address
♦ branch telephone number

☑ Users should be able to sort the search results in ascending order of mileage and ascending order of price.

Functional requirements

☑ The relational database will have two tables; one for car details and one for branch details:

♦ Each table will require a suitable primary key field.
♦ A foreign key will be used to link the two tables.

☑ Additional fields will be needed for:

♦ car make
♦ car model
♦ car colour
♦ car fuel — diesel, petrol or electric
♦ car price
♦ car mileage
♦ branch address
♦ branch telephone number

☑ Simple and complex queries will be used to search the database.
☑ A complex sort will be used to order the query results.

Administrative information

Published: May 2023 (version 3.0)

History of changes to course specification

Version	Description of change	Date
1.1	Included information on the Core Skill 'Information and communication technology at SCQF level 5' in the course overview section.	May 2017
2.0	Course support notes added as appendix.	September 2017
2.1	Changed 'developing skills in computer programming' to 'demonstrating skills in computer programming' in the 'Course assessment structure: assignment' section. Amended the 'Skills, knowledge and understanding', 'Developing skills, knowledge and understanding', and 'Resources to support the National 5 Computing Science course' sections to reflect the requirements of the new General Data Protection Regulation (GDPR).	May 2018
2.2	Amended the 'Skills, knowledge and understanding for the course assessment' and 'Resources to support the National 5 Computing Science course' sections from EU GDPR to UK GDPR.	August 2021
3.0	Amendments to the 'Course assessment overview' section, 'Course Assessment' section and 'Course support notes' to reflect the option of assessing DDD or WDD. This covers changes to the total marks and durations, and information on the structure of both the question paper and the assignment.	May 2023

This course specification may be reproduced in whole or in part for educational purposes provided that no profit is derived from reproduction and that, if reproduced in part, the source is acknowledged. Additional copies of this course specification can be downloaded from SQA's website at www.sqa.org.uk.

Note: You are advised to check SQA's website to ensure you are using the most up-to-date version of the course specification.

© Scottish Qualifications Authority 2017, 2018, 2021, 2023