



Higher Computing Science

Course code:	C816 76
Course assessment code:	X816 76
SCQF:	level 6 (24 SCQF credit points)
Valid from:	session 2023–24

This document 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.

The information in this publication may be reproduced in support of SQA qualifications only on a non-commercial basis. If it is reproduced, SQA must be clearly acknowledged as the source. If it is to be reproduced for any other purpose, written permission must be obtained from permissions@sqa.org.uk.

Where this publication includes materials from sources other than SQA (secondary copyright) this material must only be reproduced for the purposes of instruction in an educational establishment. If it is to be reproduced for any other purpose, it is the user's responsibility to obtain the necessary copyright clearance. The acknowledgements page lists sources of copyright items that are not owned by SQA.

This edition: May 2023 (version 3.0)

© Scottish Qualifications Authority 2018, 2020, 2021, 2023

Contents

Course overview	1
Course rationale	2
Purpose and aims	2
Who is this course for?	2
Course content	3
Skills, knowledge and understanding	4
Skills for learning, skills for life and skills for work	13
Course assessment	14
Course assessment structure: question paper	14
Course assessment structure: assignment	17
Grading	19
Equality and inclusion	20
Further information	21
Appendix: course support notes	22
Introduction	22
Developing skills, knowledge and understanding	22
Approaches to learning and teaching	22
Preparing for course assessment	45
Developing skills for learning, skills for life and skills for work	50
Resources to support the Higher Computing Science course	51
Appendix 1: development methodologies (SDD)	65
Appendix 2: analysis (SDD)	67
Appendix 3: design techniques (SDD)	69
Appendix 4: evaluation (SDD)	72
Appendix 5: floating-point representation (CS)	74
Appendix 6: computer structure (CS)	76
Appendix 7: environmental impact (CS)	78
Appendix 8: analysis (DDD)	79
Appendix 9: design (DDD)	82
Appendix 10: design of solution to database queries (DDD)	91
Appendix 11: SQL (DDD)	96
Appendix 12: design (WDD)	105

Appendix 13: Cascading Style Sheets (CSS) — controlling appearance and positioning (WDD)	110
Appendix 14: Cascading Style Sheets (CSS) — horizontal navigation bar (WDD)	118
Appendix 15: HTML — page layout (WDD)	121
Appendix 16: HTML — forms (WDD)	125
Appendix 17: JavaScript (WDD)	131
Appendix 18: testing (WDD)	138
Copyright acknowledgements	140

Course overview

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

The course assessment has two components.

Component	Marks	Duration
Question paper	80	2 hours
Assignment	40	see 'Course assessment' section

Recommended entry	Progression
<p>Entry to this course is at the discretion of the centre.</p> <p>Candidates should have achieved the National 5 Computing Science course or equivalent qualifications and/or experience prior to starting this course.</p>	<ul style="list-style-type: none">◆ 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.

Course rationale

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

This course highlights the central role of computing professionals as problem-solvers and designers, and the far-reaching impact of information technology on our environment and society.

It provides candidates with an understanding of the technologies and develops a wide range of practical skills that underpin our modern, digital world. The course also builds awareness of the importance of computing in meeting our needs today and for the future, in many fields including science, education, business and industry.

Purpose and aims

The course introduces candidates to an advanced range of computational processes, where they learn to apply a rigorous approach to the design and development process across a variety of contemporary contexts. They also gain an awareness of the important role that computing professionals play in meeting the needs of society today and for the future.

The course enables candidates to:

- ◆ develop and apply aspects of computational thinking in a range of contemporary contexts
- ◆ apply knowledge and understanding of advanced concepts and processes in computing science
- ◆ apply skills and knowledge in analysis, design, implementation, testing and evaluation to a range of digital solutions with some complex aspects
- ◆ communicate advanced computing concepts and explain computational behaviour clearly and concisely, using appropriate terminology
- ◆ develop awareness of current trends in computing technologies and their impact in transforming and influencing our environment and society

Who is this course for?

The course is suitable for candidates interested in exploring the role and impact of contemporary computing technologies. It provides an insight into the challenge, excitement and rewards found in these areas.

Course content

The course has four areas of study:

Software design and development

Candidates develop knowledge and understanding of advanced concepts and practical problem-solving skills in software design and development. They do this by using appropriate modular software development environments. Candidates develop modular programming and computational-thinking skills by analysing, designing, implementing, testing, and evaluating practical solutions and explaining how these programs work. They use their knowledge of data types and constructs to create efficient programs to solve advanced problems.

Computer systems

Candidates develop their understanding of how data and instructions are stored in binary form and factors affecting system performance. They gain an awareness of the environmental impact of intelligent systems, as well as the security risks, precautions and laws that can protect computer systems.

Database design and development

Candidates develop knowledge, understanding and advanced practical problem-solving skills in database design and development. They do this through a range of practical tasks, using a minimum of three linked tables and implemented in SQL. Candidates apply computational-thinking skills to analyse, design, implement, test, and evaluate practical solutions, using a range of development tools. Candidates apply interpretation skills to tasks involving some complex features in both familiar and new contexts.

Web design and development

Candidates develop knowledge, understanding and advanced practical problem-solving skills in web design and development. They do this through a range of practical and investigative tasks. Candidates apply computational-thinking skills to analyse, design, implement, test, and evaluate practical solutions to web-based problems, using a range of development tools including HTML, Cascading Style Sheets (CSS) and JavaScript. Candidates apply interpretation skills to tasks involving some complex features in both familiar and new contexts.

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 computational thinking to understand problems across a range of contexts
- ◆ analysing problems with some complex aspects within computing science across a range of contemporary contexts
- ◆ designing, implementing, testing and evaluating digital solutions (including computer programs) to problems with some complex aspects 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 advanced concepts related to software design and development, and information system design and development, clearly and concisely, using appropriate terminology
- ◆ understanding and evaluating the legal and environmental impact of contemporary computing 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 compare the development methodologies: <ul style="list-style-type: none"> ◆ iterative development process ◆ agile methodologies
Analysis	Identify the: <ul style="list-style-type: none"> ◆ purpose ◆ scope ◆ boundaries ◆ functional requirements of a problem that relates to the design and implementation at this level, in terms of: <ul style="list-style-type: none"> ◆ inputs ◆ processes ◆ outputs

Software design and development	
Design	<p>Identify the data types and structures required for a problem that relates to the implementation at this level.</p> <p>Read and understand designs of solutions to problems at this level, using the following design techniques:</p> <ul style="list-style-type: none"> ◆ structure diagrams ◆ pseudocode <p>Exemplify and implement efficient design solutions to a problem, using a recognised design technique, showing:</p> <ul style="list-style-type: none"> ◆ top level design ◆ the data flow ◆ refinements <p>Describe, exemplify and implement user-interface design, in terms of input and output, using a wireframe.</p>
Implementation (data types and structures)	<p>Describe, exemplify and implement appropriately the following structures:</p> <ul style="list-style-type: none"> ◆ parallel 1D arrays ◆ records ◆ arrays of records
Implementation (computational constructs)	<p>Describe, exemplify and implement the appropriate constructs in a procedural high-level (textual) language:</p> <ul style="list-style-type: none"> ◆ parameter passing (formal and actual) ◆ the scope of local and global variables ◆ sub-programs/routines, defined by their name and arguments (inputs and outputs): <ul style="list-style-type: none"> — functions — procedures ◆ pre-defined functions (with parameters): <ul style="list-style-type: none"> — to create substrings — to convert from character to ASCII and vice versa — to convert floating-point numbers to integers — modulus ◆ file handling: <ul style="list-style-type: none"> — sequential CSV and txt files (open, create, read, write, close) <p>Read and explain code that makes use of the above constructs.</p>

Software design and development	
Implementation (algorithm specification)	<p>Describe, exemplify and implement standard algorithms using 1D arrays or arrays of records:</p> <ul style="list-style-type: none"> ◆ linear search ◆ find minimum and maximum ◆ count occurrences
Testing	<p>Describe, exemplify and implement a comprehensive final test plan to show that the functional requirements are met.</p> <p>Identify syntax, execution, and logic errors at this level.</p> <p>Describe and exemplify debugging techniques:</p> <ul style="list-style-type: none"> ◆ dry runs ◆ trace tables/tools ◆ breakpoints ◆ watchpoints
Evaluation	<p>Describe, identify and exemplify the evaluation of a solution in terms of:</p> <ul style="list-style-type: none"> ◆ fitness for purpose ◆ efficient use of coding constructs ◆ usability ◆ maintainability ◆ robustness

Computer systems	
Data representation	<p>Describe and exemplify the use of binary to represent positive and negative integers using two's complement, including the range of numbers that can be represented using a fixed number of bits.</p> <p>Conversion of two's complement numbers from binary to denary and vice versa.</p> <p>Describe and exemplify floating-point representation of positive and negative real numbers, using the terms mantissa and exponent.</p> <p>Describe the relationship between the number of bits assigned to the mantissa/exponent, and the range and precision of floating-point numbers.</p> <p>Describe Unicode used to represent characters and its advantage over extended ASCII code (8-bit) in terms of numbers of characters.</p> <p>Describe the relative advantages and disadvantages of bit-mapped graphics versus vector graphics.</p>
Computer structure	<p>Describe the concept of the fetch-execute cycle.</p> <p>Describe the factors affecting computer system performance:</p> <ul style="list-style-type: none"> ◆ number of processors (cores) ◆ width of data bus ◆ cache memory ◆ clock speed
Environmental impact	<p>Describe the environmental impact of intelligent systems:</p> <ul style="list-style-type: none"> ◆ heating systems ◆ traffic control ◆ car management systems
Security risks and precautions	<p>Describe and identify the implications for individuals and businesses of the Computer Misuse Act 1990:</p> <ul style="list-style-type: none"> ◆ unauthorised access to computer material ◆ unauthorised access with intent to commit a further offence ◆ unauthorised modification of programs or data on a computer <p>Describe and identify the security risks of:</p> <ul style="list-style-type: none"> ◆ tracking cookies ◆ DOS (Denial of Service) attacks: <ul style="list-style-type: none"> — symptoms <ul style="list-style-type: none"> ○ slow performance, inability to access

Computer systems

- effects
 - disruption to users and businesses
- costs
 - lost revenue, labour in rectifying fault
- type of fault
 - bandwidth consumption, resource starvation, Domain Name Service (DNS)
- reasons
 - financial, political, personal

Describe how encryption is used to secure transmission of data:

- ◆ use of public and private keys
- ◆ digital certificates
- ◆ digital signatures

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	<p>Describe and exemplify entity-relationship diagrams with three or more entities, indicating:</p> <ul style="list-style-type: none"> ◆ entity name ◆ attributes ◆ name of relationship ◆ cardinality of relationship (one-to-one, one-to-many, many-to-many) <p>Describe and exemplify an instance using an entity-occurrence diagram.</p> <p>Describe and exemplify a compound key.</p> <p>Describe and exemplify a data dictionary with three or more entities:</p> <ul style="list-style-type: none"> ◆ entity name ◆ attribute name ◆ primary and foreign key ◆ attribute type: <ul style="list-style-type: none"> — text — number — date — time — Boolean ◆ attribute size ◆ validation: <ul style="list-style-type: none"> — presence check — restricted choice — field length — range <p>Exemplify a design of a solution to a query:</p> <ul style="list-style-type: none"> ◆ tables and queries ◆ fields ◆ search criteria ◆ sort order ◆ calculations ◆ grouping

Database design and development

Implementation	<p>Describe, exemplify and use SQL operations for pre-populated relational databases, with three or more linked tables:</p> <ul style="list-style-type: none">◆ UPDATE, SELECT, DELETE, INSERT statements making use of:<ul style="list-style-type: none">— wildcards— aggregate functions (MIN, MAX, AVG, SUM, COUNT)— computed values, alias— GROUP BY— ORDER BY— WHERE <p>Read and explain code that makes use of the above SQL.</p>
Testing	<p>Describe and exemplify testing:</p> <ul style="list-style-type: none">◆ SQL operations work correctly at this level
Evaluation	<p>Evaluate solution at this level in terms of:</p> <ul style="list-style-type: none">◆ 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	<p>Describe and exemplify the website structure of a multi-level website with a home page and two additional levels, with no more than four pages per level.</p> <p>Describe, exemplify and implement, taking into account end-user requirements and device type, an effective user-interface design (visual layout and readability) using wire-framing:</p> <ul style="list-style-type: none"> ◆ horizontal navigational bar ◆ relative horizontal and vertical positioning of the media ◆ form inputs ◆ file formats of the media (text, graphics, video, and audio) <p>Describe, exemplify and implement prototyping (low fidelity) from wireframe design at this level.</p>
Implementation (CSS)	<p>Describe, exemplify and implement efficient inline, internal and external Cascading Style Sheets (CSS) using grouping and descendant selectors to:</p> <ul style="list-style-type: none"> ◆ control appearance and positioning: <ul style="list-style-type: none"> — display (block, inline, none) — float (left, right) — clear (both) — margins/padding — sizes (height, width) ◆ create horizontal navigation bars: <ul style="list-style-type: none"> — list-style-type:none — hover <p>Read and explain code that makes use of the above CSS.</p>
Implementation (HTML)	<p>Describe, exemplify and implement HTML code:</p> <ul style="list-style-type: none"> ◆ nav ◆ header ◆ footer ◆ section ◆ main ◆ form ◆ id attribute

Web design and development	
	<p>Describe, exemplify and implement form elements:</p> <ul style="list-style-type: none"> ◆ form element: input <ul style="list-style-type: none"> — text — number — textarea — radio — submit ◆ form element: select <p>Describe, exemplify and implement form data validation:</p> <ul style="list-style-type: none"> ◆ length ◆ presence ◆ range <p>Read and explain code that makes use of the above HTML.</p>
Implementation (JavaScript)	<p>Describe, exemplify and implement coding of JavaScript functions related to mouse events:</p> <ul style="list-style-type: none"> ◆ onmouseover ◆ onmouseout ◆ onclick
Testing	<p>Describe, exemplify and implement usability testing using personas, test cases and scenarios based on low-fidelity prototypes.</p> <p>Describe and exemplify testing:</p> <ul style="list-style-type: none"> ◆ input validation ◆ navigational bar works ◆ media content displays correctly <p>Describe and exemplify compatibility testing:</p> <ul style="list-style-type: none"> ◆ device type: <ul style="list-style-type: none"> — tablet, smartphone, desktop ◆ browser
Evaluation	<p>Evaluate solution at this level in terms of:</p> <ul style="list-style-type: none"> ◆ fitness for purpose ◆ usability

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, and can be found on the SCQF website.

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

You must build these skills into the course at an appropriate level, where there are suitable opportunities.

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 knowledge and skills developed through the course to:

- ◆ solve appropriately challenging, practical computing science problems
- ◆ answer appropriately challenging questions in computing science contexts

Course assessment structure: question paper

Question paper 80 marks

The question paper gives candidates the opportunity to demonstrate their ability to:

- ◆ apply computational thinking to understand problems, across a range of contexts
- ◆ analyse computing science problems with some complex aspects, across a range of contemporary contexts
- ◆ design, implement, test and evaluate digital solutions (including computer programs) to problems, across a range of contemporary contexts
- ◆ communicate how a program works in technical detail
- ◆ communicate understanding of advanced concepts related to computing science clearly and concisely, using appropriate terminology
- ◆ understand the legal and environmental impact of contemporary computing technologies
- ◆ apply 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 p45 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 Higher Computing Science subject page on SQA's website.

Where candidates need 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, and conducted in centres under conditions specified for external examinations by SQA.

Candidates have 2 hours to complete the question paper.

Specimen question papers for Higher 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 their ability to:

- ◆ apply aspects of computational thinking across a range of contexts
- ◆ analyse problems within computing science across a range of contemporary contexts
- ◆ design, implement, test and evaluate digital solutions (including computer programs) to problems across a range of contemporary contexts
- ◆ demonstrate skills in computer programming
- ◆ apply 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 p45 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
- ◆ submitted to SQA for external marking

All marking is quality assured by SQA.

The specimen assessment for the course is published on SQA's website. This illustrates the standard, structure and requirements of the assessment task candidates complete. The specimen assessment task also includes 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 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, teachers and lecturers 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 or lecturers to provide support to candidates, other than to ensure that they have access to the necessary resources within the centre.

Once the assignment is complete, it must not be returned to the candidate for further work to improve their mark.

Evidence to be gathered

Candidate evidence includes program listings, screenshots or similar, as appropriate.

Volume

There is no word count.

Grading

Candidates' overall grades are 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.

- ◆ [Higher Computing Science subject page](#)
- ◆ [Assessment arrangements web page](#)
- ◆ [Building the Curriculum 3–5](#)
- ◆ [Guidance on conditions of assessment for coursework](#)
- ◆ [SQA Skills Framework: Skills for Learning, Skills for Life and Skills for Work](#)
- ◆ [Educational Research Reports](#)
- ◆ [SQA e-assessment web page](#)

The SCQF framework, level descriptors and handbook are available on the SCQF website.

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. You should read these in conjunction with this course specification and the specimen question paper and coursework.

Developing skills, knowledge and understanding

This section provides further advice and guidance about skills, knowledge and understanding that you could include in the course. You have considerable flexibility to select contexts that will stimulate and challenge candidates, offering both breadth and depth.

The 'Approaches to learning and teaching' section provides suggested experiences and activities that you can build into your delivery, to develop the skills, knowledge and understanding of the course.

Approaches to learning and teaching

The Computing Science course reflects Curriculum for Excellence values, purposes and principles, so the approaches to learning and teaching developed by individual centres should reflect these too. You should encourage candidates to participate fully in active learning and practical activities by working together, analysing, investigating, debating and evaluating topics, problems and solutions while you act increasingly as a facilitator.

You should use an appropriate balance of teaching methodologies when delivering the course. A variety of active learning approaches is encouraged, including the following:

Activity-based learning

You should balance whole-class, direct teaching opportunities with activity-based learning using practical tasks. An investigatory 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. You should support learning with appropriate practical activities, so that skills are developed simultaneously with knowledge and understanding.

Group work

Practical activities and investigations lend themselves to group work, and you should encourage this. Candidates engaged in collaborative group working strategies can capitalise on one another's knowledge, resources and skills by questioning, investigating, evaluating and presenting ideas to the group. Working as a team is a fundamental aspect of working in the IT and related industries and so should be encouraged and developed.

Problem-based learning

Problem-based learning (PBL) is another approach that can support candidates to progress through this course. This method may be best utilised at the end of a topic, where additional

challenge is required to ensure candidates are secure in their knowledge and understanding and to develop the ability to apply knowledge and skills in less familiar contexts. Learning through PBL develops skills in problem solving, decision making, investigation, creative thinking, team working and evaluation.

Computational thinking

Computational thinking is recognised as a key skill set for all 21st century candidates — whether they intend to continue with computing science or not. It involves 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 (concepts):

- ◆ **Abstraction:** seeing a problem and its solution at many levels of detail and generalising the necessary information. Abstraction allows us to represent an idea or a process in general terms (for example variables) and use it 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 we can clearly explain a process to another person — or to a computer. Decomposing a problem frequently leads to pattern recognition and generalisation/abstraction, and ultimately the ability to design an algorithm.
- ◆ **Pattern recognition:** the ability to notice similarities or common differences that will help us make predictions or lead us to shortcuts. Pattern recognition is frequently the basis for solving problems and designing algorithms.
- ◆ **Generalisation:** realising that we can use a solution to one problem to solve a whole range of related problems.

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

Computational thinking can be a component of many subjects; computing science delivers this particularly well. You 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 interest and curiosity should be a prime objective when teaching 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 support teacher-led approaches.

Blending assessment activities with learning activities throughout the course can support learning, 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
- ◆ boosting confidence by providing supportive feedback

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

Meeting the needs of all candidates

Within any class, each candidate has individual strengths and areas for improvement. If there are candidates capable of achieving a higher level in some aspects of the course, you should give them the opportunity to do so, where possible.

Where there are candidates who are struggling to achieve Higher level in some aspects of the course, you should provide opportunities for additional or peer support. You can do this by allowing a more able candidate to take on a tutor-type role and assisting other candidates to develop and reinforce their understanding of a particular topic.

When delivering this course to a group of candidates where some are working towards National 5 and others Higher, you may find it useful to identify activities covering common knowledge and skills for all candidates, and then provide additional activities for Higher candidates.

Where Higher candidates have studied National 5 in a previous year, it is important that you provide them with new and different contexts for learning to avoid demotivation. For example, candidates could work in a different type of development environment or language at Higher than they did for National 5.

Advice on distribution of time

The notional length of time for candidates to complete the course is 160 hours, although they may need to contribute some of their own time in addition to the programmed learning time.

You can decide how to distribute the time, depending on the prior learning of your candidates. You should allocate time for preparation for the question paper and 6 hours for the course assignment.

Suggested learning activities

You can decide the sequence of delivery for the four areas of study:

Software design and development

You are encouraged to use an investigatory approach, with candidates actively involved in developing their skills, knowledge and understanding of a range of software development problems and solutions.

◆ **Development methodologies:**

- Working in groups, candidates could discuss how to carry out software projects using an agile methodology, compared to an iterative development process.

◆ **Analysis:**

- Working in groups, candidates could analyse a number of problems and decide purpose, scope, boundaries and functional requirements.

◆ **Design:**

- You could present candidates with a variety of completed top level designs, and ask them to complete the data flow and the refinements.
- Ask candidates to solve complex problems using their chosen design technique, discuss the differences and decide which solutions are more efficient.
- Candidates could then design user interfaces using wireframes.

◆ **Implementation:**

- You could provide candidates with working programs that demonstrate sub-programs, user-defined functions, parameter passing, sequential file operations and scope, local and global variables.
- Ask candidates to identify and explain sections of code from within these programs.
- Using the pre-defined functions stated in the course content, candidates could tackle a number of problems.
- Using programs created in National 5, ask candidates to think how they could use their knowledge of parallel 1D-arrays, records and arrays of records to implement them using the new data structures.
- Working in groups, ask candidates to write code from designs provided in pseudocode or structure diagrams. This would help them implement the data flow design using procedures and parameter passing.
- Using a range of working programs that use a variety of standard algorithms, ask candidates to interpret and explain what is happening in the code. This would help them develop their own modular programs that make use of these constructs and standard algorithms.

◆ **Testing:**

- Using a variety of programs, ask candidates to create comprehensive final test plans to show that the functional requirements were met and to test them to check whether they work. It would be useful to use a number of programs that had logic and syntax errors, to show the benefits of testing.
- You could demonstrate debugging techniques, for example dry runs, trace tables, breakpoints and watchpoints, to show how they can help programmers find errors within their code.

◆ **Evaluation:**

- In groups, ask candidates to evaluate completed programs in terms of efficient use of coding constructs, fitness for purpose, usability, maintainability and robustness. A mixture of efficient and non-efficient programs would help demonstrate the benefits of evaluation.

Computer systems

◆ **Data representation:**

- You could describe how computers store negative integers using two's complement.
- Ask candidates to complete exercises to convert two's complement numbers from binary to denary and vice versa, and the range of numbers that can be represented using a fixed number of bits.
- Using different exercises, candidates could demonstrate understanding of the advantages of Unicode over ASCII, and the advantages and disadvantages of bit-mapped versus vector graphics.

◆ **Computer structure:**

- Candidates could research ways to improve computer performance and discuss the main points with the class.
- You could discuss the fetch-execute cycle and relate it to the programs the candidates have been writing.

◆ **Environmental impact:**

- Working in groups, candidates could investigate the environmental impact of intelligent systems in heating systems, traffic control and car management systems and then present their findings to the class.

◆ **Security risks and precautions:**

- Candidates could research instances when the Computer Misuse Act (1990) was breached and identify the implications for individuals and businesses.
- Working in teams, candidates could research tracking cookies and Denial of Service (DOS) attacks and present their findings to the class.
- You could explain the many ways that encryption can be used to securely transmit data and discuss how to ensure communications are secure.

Database design and development

◆ **Analysis:**

- Working in groups (with some candidates being database developers and others being clients), the developers could interview the clients and create the end-user and functional requirements for the database problem.

◆ **Design:**

- You could demonstrate a completed database with three or more tables, showing how the tables are connected using entity-relationship diagrams.
- Then you could explain a data dictionary, including primary keys, foreign keys, compound keys, attribute types and size, and the different types of validation.
- Candidates could complete different types of exercises to create a data dictionary from given data.

- Candidates could look at various websites to see how validation is used to stop incorrect data from being entered.

◆ **Implementation:**

- You could demonstrate SQL operations using aggregate functions, computed values and aliases.
- Candidates could then complete a number of exercises to solve problems relating to using the appropriate SQL operations.
- Using SQL code and databases, ask candidates to explain what the output of the code would be.

◆ **Testing and evaluation:**

- Using SQL code, candidates could test it and evaluate its fitness for purpose and accuracy of output.
- Using an incorrect SQL operation, along with correct expected output, ask candidates to identify how to correct the SQL statement in order to produce the expected output.

Web design and development

◆ **Analysis:**

- Working in groups (with some being web developers and others being clients), the developers could interview the clients and create the end-user and functional requirements for the problem.

◆ **Design:**

- Candidates could use wire-framing design techniques to design website structures and pages relating to multi-level websites. These could involve horizontal navigation bars and forms.
- Using the completed website designs, candidates could create low-fidelity prototypes to test their effectiveness.

◆ **Implementation:**

- Using HTML, Cascading Style Sheets (CSS) and JavaScript code with the web pages, candidates could explain which parts of the code relate to the web page.
- Using completed HTML and CSS files, ask candidates to edit the CSS code to create different appearances and positioning.
- Using some form design, ask candidates to implement using HTML.

◆ **Testing and evaluation:**

- Candidates could create test tables to check the usability and fitness for purpose of a number of websites.
- Candidates could create personas, test cases and scenarios, and implement usability testing on low-fidelity prototypes produced by others.
- Ask candidates to discuss the types of compatibility testing that website creators have to carry out before making their websites live.

Resources

The following is a summary of suggested resources:

- ◆ 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)

You should find that existing hardware and software within the computing science classroom provide all that is required to deliver the course.

Some suggested specific online resources:

[date accessed May 2018]

◆ Software design and development

www.java.com
www.python.org
www.codecademy.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.codecademy.com
www.tutorialspoint.com/sql
www.sqlcourse.com
Apex.oracle.com/en

◆ Web design and development

www.w3schools.com
www.codecademy.com
html.net/tutorials
www.khanacademy.org
pencil.evolus.vn
balsamiq.com
resources.infosecinstitute.com/prototyping
Goggles.mozilla.org
http://hackasaurus.toolness.org

Some suggested software development environments

For this course, you can use any software development environment. You should base your decision on the suitability of the chosen environment to support the delivery of the mandatory content of the course.

Possible examples of software development environments that might be suitable:

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

Comparison of National 5 and Higher

The following table shows the relationship between the mandatory National 5 and Higher knowledge and understanding. You may find it useful for:

- ◆ designing and planning learning activities for multi-level National 5/Higher classes
- ◆ ensuring seamless progression between levels
- ◆ identifying important prior learning for candidates at Higher

Software design and development		
	National 5	Higher
Development methodologies	Describe and implement the phases of an iterative development process: analysis, design, implementation, testing, documentation, and evaluation, within general programming problem solving.	Describe and compare the development methodologies: <ul style="list-style-type: none"> ◆ iterative development process ◆ agile methodologies
Analysis	Identify the purpose and functional requirements of a problem that relates to the design and implementation at this level, in terms of: <ul style="list-style-type: none"> ◆ inputs ◆ processes ◆ outputs 	Identify the: <ul style="list-style-type: none"> ◆ purpose ◆ scope ◆ boundaries ◆ functional requirements of a problem that relates to the design and implementation at this level, in terms of: <ul style="list-style-type: none"> ◆ inputs ◆ processes ◆ outputs

Software design and development		
	National 5	Higher
Design	<p>Identify the data types and structures required for a problem that relates to the implementation at this level, as listed below.</p> <p>Describe, identify, and be able to read and understand:</p> <ul style="list-style-type: none"> ◆ structure diagrams ◆ flowcharts ◆ pseudocode <p>Exemplify and implement one of the above design techniques to design efficient solutions to a problem.</p> <p>Describe, exemplify, and implement user-interface design, in terms of input and output, using a wireframe.</p>	<p>Identify the data types and structures required for a problem that relates to the implementation at this level.</p> <p>Read and understand designs of solutions to problems at this level, using the following design techniques:</p> <ul style="list-style-type: none"> ◆ structure diagrams ◆ pseudocode <p>Exemplify and implement efficient design solutions to a problem, using a recognised design technique, showing:</p> <ul style="list-style-type: none"> ◆ top level design ◆ the data flow ◆ refinements <p>Describe, exemplify, and implement user-interface design, in terms of input and output, using a wireframe.</p>
Implementation (data types and structures)	<p>Describe, exemplify, and implement appropriately the following data types and structures:</p> <ul style="list-style-type: none"> ◆ character ◆ string ◆ numeric (integer and real) ◆ Boolean ◆ 1D arrays 	<p>Describe, exemplify and implement appropriately the following structures:</p> <ul style="list-style-type: none"> ◆ parallel 1D arrays ◆ records ◆ arrays of records

Software design and development		
	National 5	Higher
Implementation (computational constructs)	<p>Describe, exemplify, and implement the appropriate constructs in a high-level (textual) language:</p> <ul style="list-style-type: none"> ◆ 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 ◆ pre-defined functions (with parameters): <ul style="list-style-type: none"> — random — round — length <p>Read and explain code that makes use of the above constructs.</p>	<p>Describe, exemplify and implement the appropriate constructs in a procedural high-level (textual) language:</p> <ul style="list-style-type: none"> ◆ parameter passing (formal and actual) ◆ the scope of local and global variables ◆ sub-programs/routines, defined by their name and arguments (inputs and outputs): <ul style="list-style-type: none"> — functions — procedures ◆ pre-defined functions (with parameters): <ul style="list-style-type: none"> — to create substrings — to convert from character to ASCII and vice versa — to convert floating-point numbers to integers — modulus ◆ file handling: <ul style="list-style-type: none"> — sequential CSV and txt files (open, create, read, write, close) <p>Read and explain code that makes use of the above constructs.</p>

Software design and development		
	National 5	Higher
Implementation (algorithm specification)	<p>Describe, exemplify, and implement standard algorithms:</p> <ul style="list-style-type: none"> ◆ input validation ◆ running total within loop ◆ traversing a 1-D array 	<p>Describe, exemplify and implement standard algorithms using 1D arrays or arrays of records:</p> <ul style="list-style-type: none"> ◆ linear search ◆ find minimum and maximum ◆ count occurrences
Testing	<p>Describe, identify, exemplify, and implement normal, extreme, and exceptional test data for a specific problem, using a test table.</p> <p>Describe and identify syntax, execution, and logic errors.</p>	<p>Describe, exemplify and implement a comprehensive final test plan to show that the functional requirements are met.</p> <p>Identify syntax, execution, and logic errors at this level.</p> <p>Describe and exemplify debugging techniques:</p> <ul style="list-style-type: none"> ◆ dry runs ◆ trace tables/tools ◆ breakpoints ◆ watchpoints
Evaluation	<p>Describe, identify, and exemplify the evaluation of a solution in terms of:</p> <ul style="list-style-type: none"> ◆ fitness for purpose ◆ efficient use of coding constructs ◆ robustness ◆ readability: <ul style="list-style-type: none"> — internal commentary — meaningful identifiers — indentation — white space 	<p>Describe, identify and exemplify the evaluation of a solution in terms of:</p> <ul style="list-style-type: none"> ◆ fitness for purpose ◆ efficient use of coding constructs ◆ usability ◆ maintainability ◆ robustness

Computer systems		
	National 5	Higher
Data representation	<p>Describe and exemplify the use of binary to represent positive integers.</p> <p>Describe floating-point representation of positive real numbers using the terms mantissa and exponent.</p> <p>Convert from binary to denary and vice versa.</p> <p>Describe extended ASCII code (8-bit) used to represent characters.</p> <p>Describe the vector graphics method of graphic representation for common objects:</p> <ul style="list-style-type: none"> ◆ rectangle ◆ ellipse ◆ line ◆ polygon <p>with attributes:</p> <ul style="list-style-type: none"> ◆ co-ordinates ◆ fill colour ◆ line colour <p>Describe the bit-mapped method of graphics representation.</p>	<p>Describe and exemplify the use of binary to represent positive and negative integers using two's complement, including the range of numbers that can be represented using a fixed number of bits.</p> <p>Conversion of two's complement numbers from binary to denary and vice versa.</p> <p>Describe and exemplify floating-point representation of positive and negative real numbers, using the terms mantissa and exponent.</p> <p>Describe the relationship between the number of bits assigned to the mantissa/exponent, and the range and precision of floating-point numbers.</p> <p>Describe Unicode used to represent characters and its advantage over extended ASCII code (8-bit) in terms of numbers of characters.</p> <p>Describe the relative advantages and disadvantages of bit-mapped graphics versus vector graphics.</p>

Computer systems		
	National 5	Higher
Computer structure	<p>Describe the purpose of the basic computer architecture components and how they are linked together:</p> <ul style="list-style-type: none"> ◆ processor (registers, ALU, control unit) ◆ memory locations with unique addresses ◆ buses (data and address) <p>Explain the need for interpreters and compilers to translate high-level program code to binary (machine code instructions).</p>	<p>Describe the concept of the fetch-execute cycle.</p> <p>Describe the factors affecting computer system performance:</p> <ul style="list-style-type: none"> ◆ number of processors (cores) ◆ width of data bus ◆ cache memory ◆ clock speed
Environmental impact	<p>Describe the energy use of computer systems, the implications on the environment and how these could be reduced through:</p> <ul style="list-style-type: none"> ◆ settings on monitors ◆ power down settings ◆ leaving computers on stand-by 	<p>Describe the environmental impact of intelligent systems:</p> <ul style="list-style-type: none"> ◆ heating systems ◆ traffic control ◆ car management systems

Computer systems		
	National 5	Higher
Security risks and precautions	<p>Describe the role of firewalls.</p> <p>Describe the use made of encryption in electronic communications.</p>	<p>Describe and identify the implications for individuals and businesses of the Computer Misuse Act 1990:</p> <ul style="list-style-type: none"> ◆ unauthorised access to computer material ◆ unauthorised access with intent to commit a further offence ◆ unauthorised modification of programs or data on a computer <p>Describe and identify the security risks of:</p> <ul style="list-style-type: none"> ◆ tracking cookies ◆ DOS (Denial of Service) attacks: <ul style="list-style-type: none"> — symptoms <ul style="list-style-type: none"> ○ slow performance, inability to access — effects <ul style="list-style-type: none"> ○ disruption to users and business — costs <ul style="list-style-type: none"> ○ lost revenue, labour in rectifying fault — type of fault <ul style="list-style-type: none"> ○ bandwidth consumption, resource starvation, Domain Name Service(DNS) — reasons <ul style="list-style-type: none"> ○ financial, political, personal <p>Describe how encryption is used to secure transmission of data:</p> <ul style="list-style-type: none"> ◆ use of public and private keys ◆ digital certificates ◆ digital signatures

Database design and development		
	National 5	Higher
Analysis	Identify the end-user and functional requirements of a database problem that relates to the implementation at this level.	Identify the end-user and functional requirements of a database problem that relates to the implementation at this level.
Design	<p>Describe and identify the implications for individuals and businesses of the UK General Data Protection Regulation (UK GDPR) that data must be:</p> <ul style="list-style-type: none"> ◆ 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 <p>Describe and exemplify entity-relationship diagrams with two entities indicating:</p> <ul style="list-style-type: none"> ◆ entity name ◆ attributes ◆ relationship (one-to-many) 	<p>Describe and exemplify entity-relationship diagrams with three or more entities, indicating:</p> <ul style="list-style-type: none"> ◆ entity name ◆ attributes ◆ name of relationship ◆ cardinality of relationship (one-to-one, one-to-many, many-to-many) <p>Describe and exemplify an instance using an entity-occurrence diagram.</p> <p>Describe and exemplify a compound key.</p> <p>Describe and exemplify a data dictionary with three or more entities:</p> <ul style="list-style-type: none"> ◆ entity name ◆ attribute name ◆ primary and foreign key ◆ attribute type: <ul style="list-style-type: none"> — text — number — date — time — Boolean

Database design and development		
	National 5	Higher
	<p>Describe and exemplify a data dictionary:</p> <ul style="list-style-type: none"> ◆ entity name ◆ attribute name ◆ primary and foreign key ◆ attribute type: <ul style="list-style-type: none"> — text — number — date — time — Boolean ◆ attribute size ◆ validation: <ul style="list-style-type: none"> — presence check — restricted choice — field length — range <p>Exemplify a design of a solution to the query:</p> <ul style="list-style-type: none"> ◆ multiple tables ◆ fields ◆ search criteria ◆ sort order 	<ul style="list-style-type: none"> ◆ attribute size ◆ validation: <ul style="list-style-type: none"> — presence check — restricted choice — field length — range <p>Exemplify a design of a solution to a query:</p> <ul style="list-style-type: none"> ◆ tables and queries ◆ fields ◆ search criteria ◆ sort order ◆ calculations ◆ grouping

Database design and development		
	National 5	Higher
Implementation	<p>Implement relational databases with two linked tables, to match the design with referential integrity.</p> <p>Describe, exemplify and implement SQL operations for pre-populated relational databases, with a maximum of two linked tables:</p> <ul style="list-style-type: none"> ◆ SELECT: <ul style="list-style-type: none"> — from — where: <ul style="list-style-type: none"> ○ AND, OR, <, >, = ○ order by with a maximum of two fields ◆ INSERT ◆ UPDATE ◆ DELETE ◆ equi-join between tables <p>Read and explain code that makes use of the above SQL.</p>	<p>Describe, exemplify and use SQL operations for pre-populated relational databases, with three or more linked tables:</p> <ul style="list-style-type: none"> ◆ UPDATE, SELECT, DELETE, INSERT statements making use of: <ul style="list-style-type: none"> — wildcards — aggregate functions (MIN, MAX, AVG, SUM, COUNT) — computed values, alias — GROUP BY — ORDER BY — WHERE <p>Read and explain code that makes use of the above SQL.</p>
Testing	<p>Describe and exemplify testing:</p> <ul style="list-style-type: none"> ◆ SQL operations work correctly at this level 	<p>Describe and exemplify testing:</p> <ul style="list-style-type: none"> ◆ SQL operations work correctly at this level
Evaluation	<p>Evaluate solution in terms of:</p> <ul style="list-style-type: none"> ◆ fitness for purpose ◆ accuracy of output 	<p>Evaluate solution at this level in terms of:</p> <ul style="list-style-type: none"> ◆ fitness for purpose ◆ accuracy of output

Web design and development		
	National 5	Higher
Analysis	Identify the end-user and functional requirements of a website problem that relates to the design and implementation at this level.	Identify the end-user and functional requirements of a website problem that relates to the design and implementation at this level.
Design	<p>Describe and exemplify the website structure with a home page, a maximum of four linked multimedia pages, and any necessary external links.</p> <p>Describe, exemplify and implement, taking into account end-user requirements, effective user-interface design (visual layout and readability) using wire-framing:</p> <ul style="list-style-type: none"> ◆ navigational links ◆ consistency across multiple pages ◆ relative vertical positioning of the media displayed ◆ file formats of the media (text, graphics, video, and audio) <p>Describe and identify the implications for individuals and businesses of the Copyright, Designs and Patents Act 1988 relating to:</p> <ul style="list-style-type: none"> ◆ web content (text, graphics, video, and audio) 	<p>Describe and exemplify the website structure of a multi-level website with a home page and two additional levels, with no more than four pages per level.</p> <p>Describe, exemplify and implement, taking into account end-user requirements and device type, an effective user-interface design (visual layout and readability) using wire-framing:</p> <ul style="list-style-type: none"> ◆ horizontal navigational bar ◆ relative horizontal and vertical positioning of the media ◆ form inputs ◆ file formats of the media (text, graphics, video, and audio) <p>Describe, exemplify and implement prototyping (low fidelity) from wireframe design at this level.</p>

Web design and development		
	National 5	Higher
	<p>Compare a range of standard file formats:</p> <ul style="list-style-type: none"> ◆ 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 <p>Describe the factors affecting file size and quality, relating to resolution, colour depth, and sampling rate.</p> <p>Describe the need for compression.</p> <p>Describe, exemplify and implement prototyping (low fidelity) from wireframe design at this level.</p>	
Implementation (CSS)	<p>Describe, exemplify and implement internal and external Cascading Style Sheets (CSS):</p> <ul style="list-style-type: none"> ◆ selectors, classes and IDs ◆ properties <ul style="list-style-type: none"> — text: <ul style="list-style-type: none"> ○ font (family, size) ○ color 	<p>Describe, exemplify and implement efficient inline, internal and external Cascading Style Sheets (CSS) using grouping and descendant selectors to:</p> <ul style="list-style-type: none"> ◆ control appearance and positioning: <ul style="list-style-type: none"> — display (block, inline, none) — float (left, right) — clear (both)

Web design and development		
	National 5	Higher
	<ul style="list-style-type: none"> ○ alignment — background colour <p>Read and explain code that makes use of the above CSS.</p>	<ul style="list-style-type: none"> — margins/padding — sizes (height, width) ◆ create horizontal navigation bars: <ul style="list-style-type: none"> — list-style-type:none — hover <p>Read and explain code that makes use of the above CSS.</p>
Implementation (HTML)	<p>Describe, exemplify and implement HTML code:</p> <ul style="list-style-type: none"> ◆ HTML ◆ head ◆ title ◆ body ◆ heading ◆ paragraph ◆ DIV ◆ link ◆ anchor ◆ IMG ◆ audio ◆ video ◆ lists — ol, ul and li 	<p>Describe, exemplify and implement HTML code:</p> <ul style="list-style-type: none"> ◆ nav ◆ header ◆ footer ◆ section ◆ main ◆ form ◆ id attribute <p>Describe, exemplify and implement form elements:</p> <ul style="list-style-type: none"> ◆ form element: input <ul style="list-style-type: none"> — text — number — textarea — radio

Web design and development		
	National 5	Higher
	<p>Describe and implement hyperlinks (internal and external), relative and absolute addressing.</p> <p>Read and explain code that makes use of the above HTML.</p>	<p>— submit</p> <ul style="list-style-type: none"> ◆ form element: select <p>Describe, exemplify and implement form data validation:</p> <ul style="list-style-type: none"> ◆ length ◆ presence ◆ range <p>Read and explain code that makes use of the above HTML.</p>
Implementation (JavaScript)	<p>Describe and identify JavaScript coding related to mouse events:</p> <ul style="list-style-type: none"> ◆ onmouseover ◆ onmouseout 	<p>Describe, exemplify and implement coding of JavaScript functions related to mouse events:</p> <ul style="list-style-type: none"> ◆ onmouseover ◆ onmouseout ◆ onclick
Testing	<p>Describe and exemplify testing:</p> <ul style="list-style-type: none"> ◆ matches user-interface design ◆ links and navigation work correctly ◆ media (such as text, graphics, and video) display correctly ◆ consistency 	<p>Describe, exemplify and implement usability testing using personas, test cases and scenarios based on low-fidelity prototypes.</p> <p>Describe and exemplify testing:</p> <ul style="list-style-type: none"> ◆ input validation ◆ navigational bar works ◆ media content displays correctly

Web design and development		
	National 5	Higher
		Describe and exemplify compatibility testing: <ul style="list-style-type: none"> ◆ device type: <ul style="list-style-type: none"> — tablet, smartphone, desktop ◆ browser
Evaluation	Evaluate solution in terms of: <ul style="list-style-type: none"> ◆ fitness for purpose 	Evaluate solution at this level in terms of: <ul style="list-style-type: none"> ◆ fitness for purpose ◆ usability

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, you should give candidates the opportunity to practise activities similar to those expected in the course assessment. For example, you could develop questions and tasks similar to those in the specimen question paper and specimen coursework.

You may find the following information useful:

- ◆ 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	2-7	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: 2 hours

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 questions:

- ◆ 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	2-7	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

You 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 you 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.

Skill	How to develop
Numeracy	
2.1 Number processing	Give candidates opportunities to develop their number processing skills by practising problem solving in numeric-based contexts involving, for example, multiplication, division or calculating percentages. Set problem-solving contexts where software would take decisions and vary the output based on the results of calculations.
2.3 Information handling	Develop information-handling skills by setting problem-solving contexts where candidates use data set out in tables or a graphical format as the basis for input to their programs, processing the data to produce the required output.
Employability, enterprise and citizenship	
4.2 Information and communication technology (ICT)	Throughout the course, candidates will be continually interacting with the technology around them. This should provide plenty of opportunities to extend their ICT skills.
Thinking skills	
5.3 Applying	Give candidates opportunities to analyse a wide range of problems, apply the knowledge and skills they have acquired and then test and review their solutions.
5.4 Analysing and evaluating	Develop skills in analysing and evaluating through the process of creating computer programs to solve problems and testing them.

Resources to support the Higher Computing Science course

Note: 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 Higher Computing Science' within 'Computing Science Documents'. You can access this from the following link:

glowscotland.sharepoint.com/sites/PLC/technologies/SitePages/Computing%20Science.aspx

Software design and development (SDD)		
Skills, knowledge and understanding		Exemplification/learning and teaching activities and resources
Development methodologies	Describe and compare the development methodologies: <ul style="list-style-type: none"> ◆ iterative development process ◆ agile methodologies 	See appendix 1 — development methodologies
Analysis	Identify the: <ul style="list-style-type: none"> ◆ purpose ◆ scope ◆ boundaries ◆ functional requirements of a problem that relates to the design and implementation at this level, in terms of: <ul style="list-style-type: none"> ◆ inputs ◆ processes 	See appendix 2 — software analysis

Software design and development (SDD)		
Skills, knowledge and understanding		Exemplification/learning and teaching activities and resources
	<ul style="list-style-type: none"> ◆ outputs 	
Design	<p>Identify the data types and structures required for a problem that relates to the implementation at this level.</p> <p>Read and understand designs of solutions to problems at this level, using the following design techniques:</p> <ul style="list-style-type: none"> ◆ structure diagrams ◆ pseudocode <p>Exemplify and implement efficient design solutions to a problem, using a recognised design technique showing:</p> <ul style="list-style-type: none"> ◆ top level design ◆ the data flow ◆ refinements <p>Describe, exemplify, and implement user-interface design, in terms of input and output, using a wireframe.</p>	<p>See appendix 3 — software design techniques</p> <p>Software design teaching materials</p>
Implementation (data types and structures)	<p>Describe, exemplify and implement appropriately the following structures:</p> <ul style="list-style-type: none"> ◆ parallel 1D arrays ◆ records ◆ arrays of records 	

Software design and development (SDD)		
Skills, knowledge and understanding		Exemplification/learning and teaching activities and resources
Implementation (computational constructs)	<p>Describe, exemplify, and implement the appropriate constructs in a procedural high-level (textual) language:</p> <ul style="list-style-type: none"> ◆ parameter passing (formal and actual) ◆ the scope of local and global variables ◆ sub-programs/routines, defined by their name and arguments (inputs and outputs): <ul style="list-style-type: none"> — functions — procedures ◆ pre-defined functions (with parameters) to: <ul style="list-style-type: none"> — create substrings — convert from character to ASCII and vice versa — modulus — convert floating-point numbers to integers ◆ file handling <ul style="list-style-type: none"> — sequential CSV and txt files (open, create, read, write, close) <p>Read and explain code that makes use of the above constructs.</p>	Python teaching materials
Implementation (algorithm specification)	<p>Describe, exemplify, and implement standard algorithms using 1D arrays or arrays of records:</p> <ul style="list-style-type: none"> ◆ linear search ◆ find minimum and maximum ◆ count occurrences 	

Software design and development (SDD)		
Skills, knowledge and understanding		Exemplification/learning and teaching activities and resources
Testing	<p>Describe, exemplify and implement a comprehensive final test plan to show that the functional requirements are met.</p> <p>Identify syntax, execution, and logic errors at this level.</p> <p>Describe and exemplify debugging techniques:</p> <ul style="list-style-type: none"> ◆ dry runs ◆ trace tables/tools ◆ breakpoints ◆ watchpoints 	
Evaluation	<p>Describe, identify, and exemplify the evaluation of a solution in terms of:</p> <ul style="list-style-type: none"> ◆ fitness for purpose ◆ efficient use of coding constructs ◆ usability ◆ maintainability ◆ robustness 	See appendix 4 — software evaluation
Software design and development	Overview	Homework and formative classroom assessment

Computer systems (CS)		
Skills, knowledge and understanding		Exemplification/learning and teaching activities and resources
Data representation	<p>Describe and exemplify the use of binary to represent positive and negative integers using two's complement, including the range of numbers that can be represented using a fixed number of bits.</p> <p>Conversion of two's complement numbers from binary to denary and vice versa.</p> <p>Describe and exemplify floating-point representation of positive and negative real numbers, using the terms mantissa and exponent.</p> <p>Describe the relationship between the number of bits assigned to the mantissa/exponent, and the range and precision of floating-point numbers.</p> <p>Describe Unicode used to represent characters and its advantage over extended ASCII code (8-bit) in terms of numbers of characters.</p> <p>Describe the relative advantages and disadvantages of bit-mapped graphics versus vector graphics.</p>	See appendix 5 — floating-point representation
Computer structure	<p>Describe the concept of the fetch-execute cycle.</p> <p>Describe the factors affecting computer system performance:</p> <ul style="list-style-type: none"> ◆ number of processors (cores) ◆ width of data bus ◆ cache memory ◆ clock speed 	See appendix 6 — computer structure

Computer systems (CS)		
Skills, knowledge and understanding		Exemplification/learning and teaching activities and resources
Environmental impact	<p>Describe the environmental impact of intelligent systems:</p> <ul style="list-style-type: none"> ◆ heating systems ◆ traffic control ◆ car management systems 	See appendix 7 — environmental impact
Security risks and precautions	<p>Describe and identify the implications for individuals and businesses of the Computer Misuse Act 1990:</p> <ul style="list-style-type: none"> ◆ unauthorised access to computer material ◆ unauthorised access with intent to commit a further offence ◆ unauthorised modification of programs or data on a computer <p>Describe and identify the security risks of:</p> <ul style="list-style-type: none"> ◆ tracking cookies ◆ DOS (Denial of Service) attacks: <ul style="list-style-type: none"> — symptoms <ul style="list-style-type: none"> ○ slow performance, inability to access — effects <ul style="list-style-type: none"> ○ disruption to users and business — costs <ul style="list-style-type: none"> ○ lost revenue, labour in rectifying fault — type of fault <ul style="list-style-type: none"> ○ bandwidth consumption, resource starvation, Domain Name Service(DNS) 	

Computer systems (CS)		
Skills, knowledge and understanding		Exemplification/learning and teaching activities and resources
	<ul style="list-style-type: none"> — reasons <ul style="list-style-type: none"> ○ financial, political, personal <p>Describe how encryption is used to secure transmission of data:</p> <ul style="list-style-type: none"> ◆ use of public and private keys ◆ digital certificates ◆ digital signatures 	
Computer systems	Overview	Homework and formative classroom assessment

Database design and development (DDD)		
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 8 — database analysis
Design	<p>Describe and exemplify entity-relationship diagrams with three or more entities, indicating:</p> <ul style="list-style-type: none"> ◆ entity name ◆ attributes ◆ name of relationship ◆ cardinality of relationship (one-to-one, one-to-many, many-to-many) <p>Describe and exemplify an instance using an entity-occurrence diagram.</p> <p>Describe and exemplify a compound key.</p> <p>Describe and exemplify a data dictionary with three or more entities:</p> <ul style="list-style-type: none"> ◆ entity name ◆ attribute name ◆ primary and foreign key ◆ attribute type: <ul style="list-style-type: none"> — text — number — date — time — Boolean 	<p>See appendix 9 — database design</p> <p>Database design teaching materials</p> <p>See appendix 10 — design of solution to database queries</p>

Database design and development (DDD)	
Skills, knowledge and understanding	Exemplification/learning and teaching activities and resources
<ul style="list-style-type: none"> ◆ attribute size ◆ validation: <ul style="list-style-type: none"> — presence check — restricted choice — field length — range <p>Exemplify a design of a solution to a query:</p> <ul style="list-style-type: none"> ◆ tables and queries ◆ fields ◆ search criteria ◆ sort order ◆ calculations ◆ grouping 	

Database design and development (DDD)		
Skills, knowledge and understanding		Exemplification/learning and teaching activities and resources
Implementation	<p>Describe, exemplify and use SQL operations for pre-populated relational databases, with three or more linked tables:</p> <ul style="list-style-type: none"> ◆ UPDATE, SELECT, DELETE, INSERT statements making use of: <ul style="list-style-type: none"> — wildcards — aggregate functions (MIN, MAX, AVG, SUM, COUNT) — computed values, alias — GROUP BY — ORDER BY — WHERE <p>Read and explain code that makes use of the above SQL.</p>	<p>See appendix 11 — SQL</p> <p>SQL teaching Materials</p> <p>www.w3schools.com www.sqlcourse.com www.codeacademy.com www.tutorialspoint.com/sql</p>
Testing	<p>Describe and exemplify testing:</p> <ul style="list-style-type: none"> ◆ SQL operations work correctly at this level 	
Evaluation	<p>Evaluate solution at this level in terms of:</p> <ul style="list-style-type: none"> ◆ fitness for purpose ◆ accuracy of output 	
Database design and development	Overview	Homework and formative classroom assessment

Web design and development (WDD)		
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.	
Design	<p>Describe and exemplify the website structure of a multi-level website with a home page and two additional levels, with no more than four pages per level.</p> <p>Describe, exemplify and implement, taking into account end-user requirements and device type, an effective user-interface design (visual layout and readability) using wire-framing:</p> <ul style="list-style-type: none"> ◆ horizontal navigational bar ◆ relative horizontal and vertical positioning of the media ◆ form inputs ◆ file formats of the media (text, graphics, video, and audio) <p>Describe, exemplify and implement prototyping (low fidelity) from wireframe design at this level.</p>	See appendix 12 — web design
Implementation (CSS)	<p>Describe, exemplify and implement efficient inline, internal and external Cascading Style Sheets (CSS) using grouping and descendant selectors to:</p> <ul style="list-style-type: none"> ◆ control appearance and positioning: <ul style="list-style-type: none"> — display (block, inline, none) — float (left, right) — clear (both) — margins/padding 	<p>See appendix 13 — CSS controlling appearance and positioning</p> <p>See appendix 14 — CSS horizontal navigation bars.</p> <p>Web-creation teaching materials</p> <p>HTML/CSS online teaching resources</p>

Web design and development (WDD)		
Skills, knowledge and understanding		Exemplification/learning and teaching activities and resources
	<ul style="list-style-type: none"> — sizes (height, width) ◆ create horizontal navigation bars: <ul style="list-style-type: none"> — list-style-type:none — hover <p>Read and explain code that makes use of the above CSS.</p>	<p>www.w3schools.com www.codecademy www.net.tutorials www.khanacademy.org</p>
Implementation (HTML)	<p>Describe, exemplify and implement HTML code:</p> <ul style="list-style-type: none"> ◆ nav ◆ header ◆ footer ◆ section ◆ main ◆ form ◆ id attribute <p>Describe, exemplify and implement form elements:</p> <ul style="list-style-type: none"> ◆ form element: input <ul style="list-style-type: none"> — text — number — textarea — radio — submit 	<p>See appendix 15 — HTML page layout</p> <p>Web-creation teaching materials</p> <p>See appendix 16 — HTML forms</p>

Web design and development (WDD)		
Skills, knowledge and understanding		Exemplification/learning and teaching activities and resources
	<ul style="list-style-type: none"> ◆ form element: select <p>Describe, exemplify and implement form data validation:</p> <ul style="list-style-type: none"> ◆ length ◆ presence ◆ range <p>Read and explain code that makes use of the above HTML.</p>	
Implementation (JavaScript)	<p>Describe, exemplify and implement coding of JavaScript functions related to mouse events:</p> <ul style="list-style-type: none"> ◆ onmouseover ◆ onmouseout ◆ onclick 	<p>See appendix 17 — JavaScript</p> <p>Web-creation teaching materials</p>
Testing	<p>Describe, exemplify and implement usability testing using personas, test cases and scenarios based on low-fidelity prototypes.</p> <p>Describe and exemplify testing:</p> <ul style="list-style-type: none"> ◆ input validation ◆ navigational bar works ◆ media content displays correctly <p>Description and exemplification of compatibility testing including:</p>	<p>See appendix 18 — web testing</p>

Web design and development (WDD)		
Skills, knowledge and understanding		Exemplification/learning and teaching activities and resources
	<ul style="list-style-type: none"> ◆ device type: <ul style="list-style-type: none"> — tablet, smart phone, desktop ◆ browser 	
Evaluation	Evaluate solution at this level in terms of: <ul style="list-style-type: none"> ◆ fitness for purpose ◆ usability 	
Web design and development	Overview	Homework and formative classroom assessment

Appendix 1: development methodologies (SDD)

The following table compares the iterative software development cycle (often called the waterfall model) with agile software development.

	Iterative	Agile
Client interaction	The client is heavily involved in the initial analysis stage and at the end of development, when evaluating if the software meets their needs and matches the agreed specification.	The client is involved throughout the process, giving constant feedback on prototypes of the software during development. This feedback is acted upon, quickly ensuring the software evolves throughout the project. Changing goals during the development can be positive in terms of final client satisfaction with the product.
Teamwork	Teams of analysts, programmers, testers and documenters work independently on each phase of development. Teams mainly work in isolation with some communication required between each phase.	Teams of developers communicate and collaborate, rather than teams of experts operating in isolation. During a project, fast, face-to-face communication between individuals with different skills is an important factor in progressing the project quickly.
Documentation	A detailed project specification is created at the beginning of a project. Significant time is spent during the project on design, program commentary and test plans.	While modelling solutions remains important, creating large documents that are never updated or referred to again upon completion of the project are not. Agile focuses on reducing documentation. It spends time on small cycles of coding, testing and adapting to change. Any documentation produced (for example internal commentary in code) should focus purely on progressing the project.

	Iterative	Agile
Measurement of progress	Follows a strict plan, with progress measured against timescales set at the beginning of the project.	<p>Breaks a project down into a series of short development goals (often called “sprints”). This involves cross-functional teams working on: planning, analysis, design, coding, unit testing, and acceptance testing.</p> <p>Progress is measured by the time it takes to produce prototypes or working components of the software.</p> <p>Agile focuses on delivering software as quickly as possible.</p>
Adaptive vs predictive	<p>A predictive methodology, focusing on analysing and planning the future in detail and catering for known risks.</p> <p>Predictive methods rely on effective early phase analysis and if this goes very wrong, the project may have difficulty changing direction.</p> <p>Predictive teams often institute a change control board to ensure they consider only the most valuable changes.</p>	<p>An adaptive methodology, focusing on adapting quickly to changing realities. When the needs of a project change, an adaptive team changes as well.</p> <p>An adaptive team has difficulty describing exactly what they will do next week but could report on which features they plan for next month.</p> <p>The further away a date is, the vaguer an adaptive method is about what will happen on that date.</p>
Testing	Testing is carried out when the implementation phase of the project is complete.	There is no recognised testing phase, as testing is carried out in conjunction with programming.

Appendix 2: analysis (SDD)

This is the start of the software development process and defines the extent of the software task. This is called the software specification. It is often the basis of a legal contract between the client (customer) and the software company writing the software.

Your analysis should include the following:

- ◆ Purpose: a general description of the purpose of the software.
- ◆ Scope: a list of the deliverables that the project will hand over to the client and/or end-user, eg design, completed program, test plan, test results and evaluation report. It can also include any time limits for the project.
- ◆ Boundaries: the limits that help to define what is in the project and what is not. It can also clarify any assumptions made by the software developers regarding the client's requirements.
- ◆ Functional requirements: the features and functions that must be delivered by the system in terms of inputs, processes and outputs.

Exemplar

Analysis	
Purpose	The purpose of this program is to take 20 pupil names, their prelim marks and their assignment marks from a file. Calculate the percentage, and then find and display the name and percentage of the pupil with the highest percentage.
Scope	<p>This development involves creating a modular program. The deliverables include:</p> <ul style="list-style-type: none">◆ detailed design of the program structure◆ test plan with completed test data table◆ working program◆ results of testing◆ evaluation report <p>This development work must be completed within 4 hours.</p>
Boundaries	<p>The program will read the pupil data (name, prelim mark and assignment mark) for 20 pupils from a sequential file. The data is accurate, so there is no need to implement input validation.</p> <p>The pupil with the top mark will be the pupil who has the highest percentage. The only output needed is the name and percentage of the pupil with the highest percentage.</p>

Analysis

Functional requirements	<p>These are defined in terms of the inputs, processes and outputs detailed below. All inputs are imported from a sequential file and all outputs displayed on the screen. The program is activated by double clicking on the file icon and then selecting “Run” from the menu. Each process should be a separate procedure or function that is called from the main program.</p> <p>Inputs: Pupil name Prelim mark Assignment mark</p> <p>Processes: Calculate the percentage for each pupil Find the name and percentage of the pupil with the highest percentage</p> <p>Output: Name of the pupil with the highest percentage The highest percentage</p>
-------------------------	---

Appendix 3: design techniques (SDD)

Pseudocode

When using pseudocode to design efficient solutions to a modular problem, you must include the following:

- ◆ Top level design — the major steps of the design. In the example below, numbered from 1 to 4.
- ◆ Data flow — shows the information that must flow In or Out from the sub-programs. In the example below, written to the right of the top level design.
- ◆ Refinements — break down the design from the top level when required. In the example below, numbered as a sub-number of the top level.

The following design is for a program that will read the name, prelim mark and coursework mark for a class of 20 pupils from a file. It will calculate a percentage from each of their prelim marks and coursework marks added together. It will then display the name of the pupil with the highest percentage and their percentage.

- | | | |
|---|--------------------------------------|---|
| 1 | Get results | (OUT: pupil name(), prelim mark(), course mark()) |
| 2 | Calculate percentages | (IN: prelim mark(), course mark())
OUT: percentage() |
| 3 | Find position of pupil with top mark | (IN: percentage() OUT: top position) |
| 4 | Display pupil with top mark | (IN: pupil name(), top position) |

1.1 Open marks file

1.2 Start fixed loop for each pupil

1.3 Get pupil name()

1.4 Get prelim mark()

1.5 Get course mark()

1.6 End fixed loop

1.7 Close marks file

2.1 Start fixed loop for each pupil

2.2 percentage() equals (prelim mark() + course mark()) divided by 1.5

2.3 End fixed loop

3.1 top position equals first position

3.2 Start fixed loop from second pupil

3.3 If percentage() is greater than current top percentage Then

3.4 set position as new top position

3.5 End If

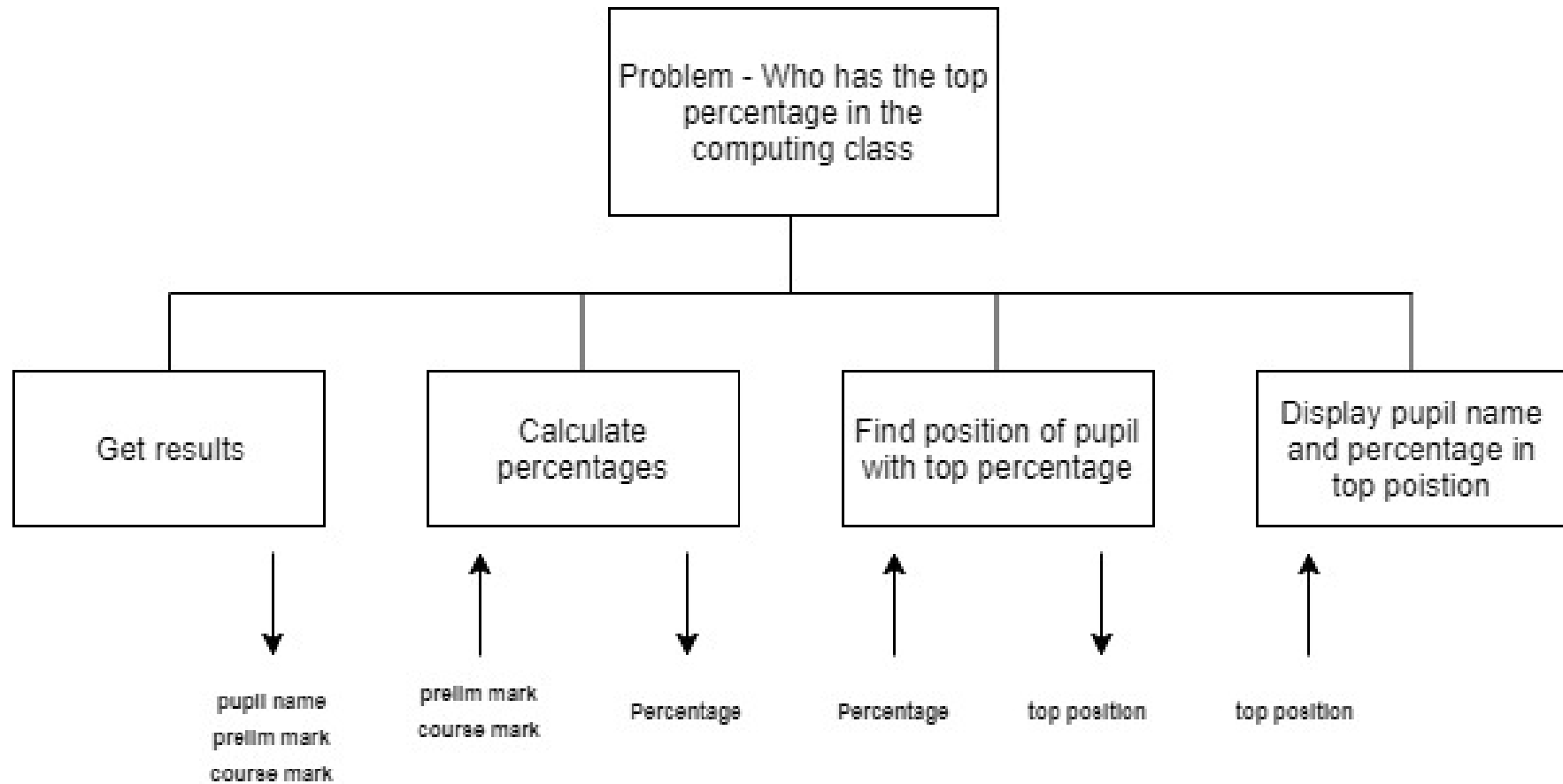
3.6 End fixed loop

4.1 Display "Top pupil is", pupil name(top position), "with", percentage(top position), "percent"

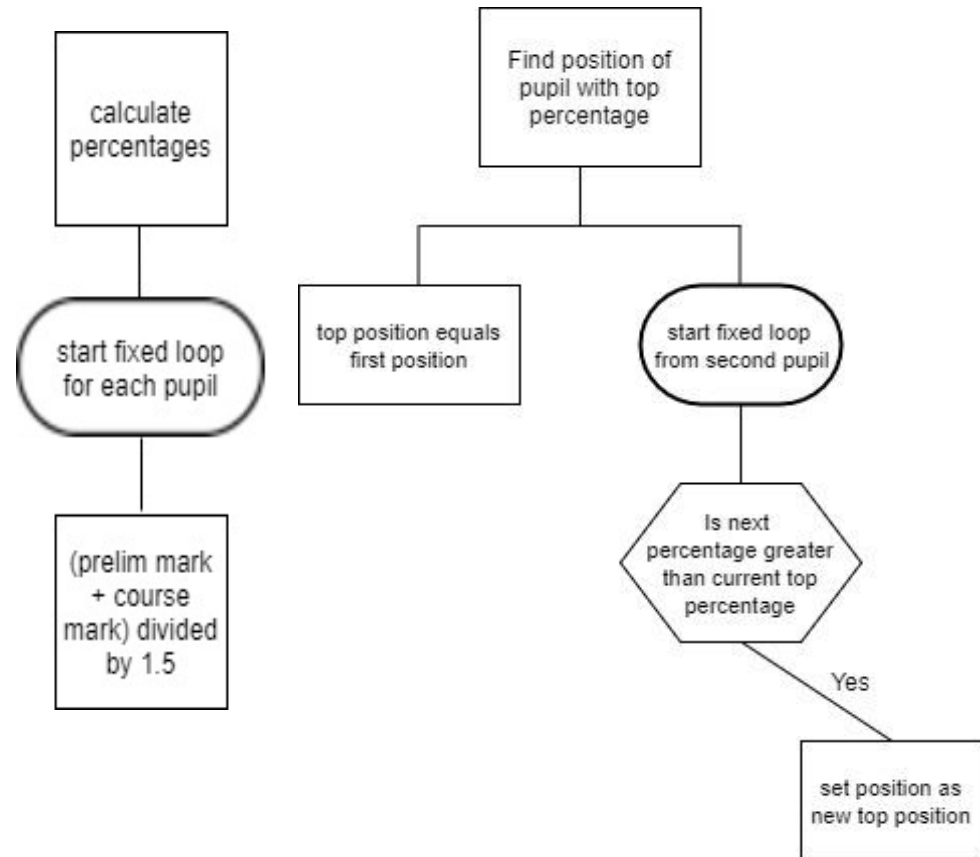
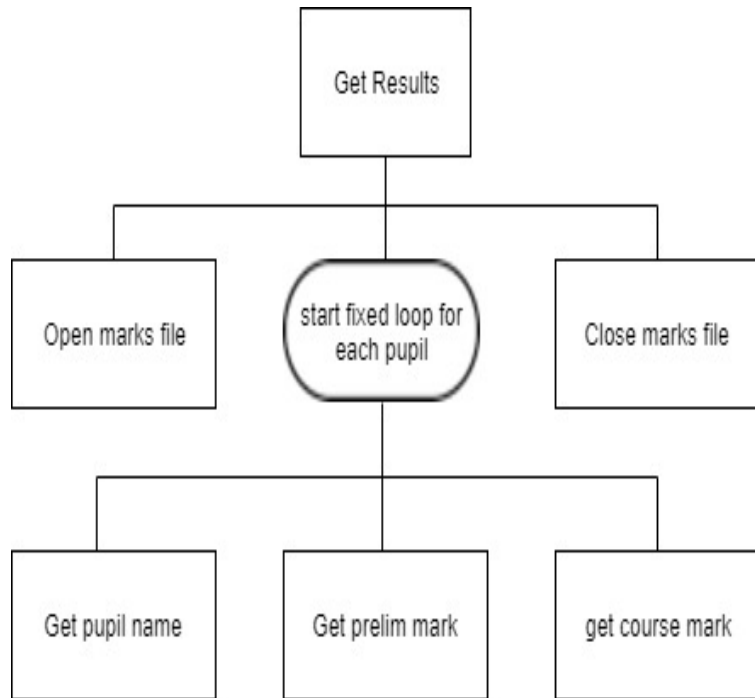
Structure diagrams

The following structure diagram solves the same problem as the pseudocode:

- ◆ Top level design — the major steps of the design.
- ◆ Data flow — shows the information that must flow In or Out from the sub-programs. In the example below, written underneath the top level design with an arrow showing whether they are In or Out.



- ◆ Refinements — break down the design from the top level into smaller steps. They can be shown separately from the top level design or below the top level design.



Appendix 4: evaluation (SDD)

This is an objective review of the software to establish whether it meets the required criteria.

Fitness for purpose

This reflects whether the software carries out all the tasks required of the software specification.

Your evaluation should identify any discrepancies between the software specification and the completed software.

Efficient use of coding constructs

This reflects whether the software writers have used their knowledge of constructs to help them create efficient code. For example using:

- ◆ suitable data types or structures
- ◆ conditional or fixed loops
- ◆ arrays
- ◆ nested selection
- ◆ procedures or functions with parameter passing

Your evaluation should identify where your coding has been efficient.

Usability

This reflects how intuitive the software is from a user's perspective and should include:

- ◆ the general user interface
- ◆ the user prompts
- ◆ the screen layout
- ◆ any help screens

Your evaluation should identify features of the software that have enhanced usability for the user.

Maintainability

This reflects how easy it is to alter the software. The factors affecting maintainability include:

- ◆ readability of the code — made easier by using meaningful variable names, comments, indentation and whitespace
- ◆ amount of modularity — using functions and procedures effectively

Your evaluation should identify how your code helps with the maintainability of the software.

Robustness

This reflects how well the software copes with errors during execution including:

- ◆ exceptional data, eg the computer crashing if “out of range”
- ◆ incorrect data entered

Your evaluation should reflect the testing that has been undertaken to meet the specification, as well as to demonstrate some degree of robustness.

Appendix 5: floating-point representation (CS)

There are many ways to represent floating-point numbers, with the majority beyond the requirements of the Higher Computing Science course. The course includes a simplified version of floating-point representation, which takes its basis from the BCS Glossary of Computing, 14th edition, 2016 (page 267).

Real numbers are stored in a computer as floating-point numbers. Floating point is a form of representation where numbers are expressed as a binary or decimal fractional value called the mantissa, together with an integer exponent. You can represent any number in any number base, in the form:

$$m \times b^e$$

where **m** is the **mantissa**, **b** is the **base** and **e** is the **exponent**.

To convert 34 006.8 to floating-point representation involves moving the point so the number is a fractional value, and then counting how many decimal places you have moved the point, for example

$$34\ 006.8 = 0.340068 \times 10^5$$

The 5 represents the fact that the point has moved 5 places to the left. The 10 represents the base. The 5 is called the exponent and 340068 is called the mantissa.

The example above in denary is only used to illustrate the concept of floating-point representation — all exemplification is expected to be in binary.

Here is an example in binary $1010.001 = 0.1010001 \times 2^{100}$

The mantissa is 1010001 and the exponent is 100.

The mantissa and exponent are usually of a fixed size, for example 8 bit for the exponent and 16 bits for the mantissa including a sign bit. The sign bit of the mantissa is represented in binary as 0 for positive and 1 for negative. The exponent is represented in two's complement, giving a range from -128 to 127.

Fixed point	Floating point	Sign (1bit)	Mantissa (15 bit)	Exponent (8 bit)
100010.101	$0.100010101 \times 2^{111}$	0	10001010100000	0000111
-101.00011	$-0.10100011 \times 2^{11}$	1	101000110000000	0000011
0.0010101	0.10101×2^{-10}	0	101010000000000	1111110
-0.00001101	-0.1101×2^{-100}	1	110100000000000	1111100

Range and precision

The number of digits quoted in the mantissa indicates the **precision** of the number and the number of digits in the exponent is a measure of the **range of numbers** that can be stored. The number of total bits used to store both mantissa and exponent stays the same, so:

Increase mantissa, exponent must be decreased = increased precision, decreased range

Decrease mantissa, exponent must be increased = decreased precision, increased range

Appendix 6: computer structure (CS)

The fetch-execute cycle

This is the process where an instruction is retrieved from memory, decoded and then carried out.

- ◆ The processor sets up the address bus with the required address.
- ◆ The processor activates the read line on the control bus.
- ◆ An instruction is fetched from the memory location using the data bus and stored in the instruction register.
- ◆ The instruction in the instruction register is then interpreted by the decoder and carried out.

Factors affecting system performance.

A number of factors can improve the performance of a computer system. These include:

◆ **Number of processors (cores)**

This is the development of several sets of processor components in one microprocessor. A dual core processor has two separate CPU's in one chip and a quad core processor has four separate CPU's in one chip. The more cores a processor has, the more sets of instructions the processor can receive and process at the same time — this improves system performance. A dual core processor is not as fast as a single processor running at twice the speed, as it is not always possible to share some tasks equally between the cores. This reduces efficiency.

◆ **Width of data bus**

The data bus is a set of parallel wires that connects the processor with memory and other hardware devices. By increasing the data bus from 32 wires to 64 wires, the computer can transfer twice as much information at one time. Therefore, increasing the size of the data bus improves the system performance of the computer.

◆ **Cache memory**

Cache memory is a small amount of fast accessible memory, usually on the same chip as the processor. The processor checks this for data or instructions before accessing the main memory. If it finds the data or instruction, then this is termed as a cache 'hit', resulting in an improved performance. If the instruction is not present, then a cache 'miss' occurs and a slower main memory is accessed. Many computers use multiple levels of cache, with small caches backed up by larger, slower caches. Multi-level caches operate by checking the fastest cache (level 1) first. If it has a match, the processor proceeds at high speed. If it does not have a match, it checks the next fastest cache (level 2) and so on.

◆ **Clock speed**

This is the electronic unit that synchronises related components by generating pulses at a constant rate. Clock pulses are used to trigger components to take their next step. The clock rate is the frequency at which the clock generates pulses. The higher the clock rate, the faster the computer may complete a series of instructions. Different manufacturers measure the clock rate in different ways, so it is not always possible to do direct comparisons between different processor manufacturers.

Appendix 7: environmental impact (CS)

Heating systems

Smart heating systems use a variety of ways to control the amount of heat required in our homes. Using activity sensors, some smart systems learn the temperatures that you prefer in certain rooms and at what times. Monitoring the activity in rooms can mean that the smart system adjusts the heating up or down depending on whether there is unusual activity in the house. The thermostat is connected to wi-fi and can be manually controlled by using an app on your phone. This allows you to turn the heating system off if you are not going home or to turn it on so that it is at the optimum temperature if you are coming home early.

Traffic control

Vehicles are considered one of the main contributing sources of greenhouse gas. Studies in the European Union showed that transport causes 25% of all carbon dioxide emissions. Vehicles consume greater amounts of fuel when they are constantly accelerating and braking in traffic jams. The optimum speed for low fuel consumption and low emissions is between 45 and 65 miles per hour.

Intelligent transport systems use software and hardware, along with information and communications technologies, to improve the efficiency and safety of transport networks. They use a variety of information from cameras and sensors, along with control of traffic signals, to try to keep traffic moving, reducing the amount of harmful emissions. Cars with individual navigation systems use satellite information on traffic flow to guide drivers away from traffic congestion and on to more free-flowing routes.

Car management systems

A number of different car management systems are used to reduce the impact on the environment.

Start-stop systems automatically shut down the engine when the car is not moving — this reduces the amount of time the engine spends idling, reducing fuel consumption and emissions. The car automatically re-starts when the accelerator is pressed, which is most advantageous for vehicles that spend significant amounts of time waiting at traffic lights or frequently come to a stop in traffic jams.

Engine control units use sensors to ensure the engine's air/fuel ratio can be controlled very accurately, ensuring optimum fuel consumption and a reduction of carbon dioxide emissions.

Appendix 8: analysis (DDD)

During the analysis stage of database development, you should identify the following requirements:

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

Worked example

A travel agency wants to create a relational database to store details of bookings for hotels in Scottish holiday resorts. The database will allow travel agents to view details of hotels and make bookings for customers. Four separate entities are required:

- ◆ Hotel (used to store details of hotels in each resort)
- ◆ Resort (used to store details of Scottish holiday resorts)
- ◆ Customer (used to store details of customers who make holiday bookings)
- ◆ Booking (used to store details of hotel bookings)

They have appointed a developer team to carry out an analysis of the database requirements. The developers ask some of the travel agency staff about the features they would expect to see in the completed database. The following are a few of the comments made by the staff:



End-user requirements

Travel agency staff should be able to perform a range of searches to display:

- ◆ full details of any booking
- ◆ availability of hotels in a particular resort, with specified facilities (meal plan or pool)
- ◆ details of hotels in a particular type of resort
- ◆ details of hotels available for a specified star rating
- ◆ resorts that have train stations

Staff should be able to sort search results in order of ascending order of price and should be able to calculate:

- ◆ the total cost of any holiday booking
- ◆ the number of hotels within a certain price range or available on a certain start date

Functional requirements

The relational database will have four tables: Hotel, Resort, Booking and Customer.

Each table requires a suitable primary key field, with foreign keys linking the four tables.

In addition to a primary key and any necessary foreign keys, the following fields are required:

- ◆ Hotel:
 - hotel name
 - start of season date
 - check-in time
 - price per night
 - meal plan
 - swimming pool
 - star rating
- ◆ Resort
 - resort name
 - resort type
 - train station
- ◆ Customer
 - first name
 - surname
 - address
 - town
 - postcode
- ◆ Booking
 - start date
 - number in party
 - number of nights

Use the following:

- ◆ simple and complex queries to search the database
- ◆ a simple sort to order the query results
- ◆ a calculation to work out the total cost of a booking
- ◆ an aggregate function to work out the number of 4-star hotels located in resorts that have a train station

Appendix 9: design (DDD)

Entity-relationship modelling

In the development of a complex database system, entity-relationship modelling is used to plan the structure of the database. In entity-relationship modelling, specialist terminology is used to define each component of the model. This terminology includes:

- ◆ entity
- ◆ entity occurrence
- ◆ attribute
- ◆ primary key
- ◆ foreign key
- ◆ compound key
- ◆ relationship
- ◆ cardinality
- ◆ entity-occurrence diagram
- ◆ entity-relationship diagram

For a definition of each of these terms, see below:

Entity

An entity is a person, place, thing, event or concept of interest to the business or organisation about which data is to be stored. For example, in a school, possible entities might be Student, Teacher, Class and Subject.

Entity occurrence

A specific example of an entity is called an instance or entity occurrence. For example, John Smith, Mary McLeod and Omar Shaheed are all entity occurrences found in the Student entity; English, Computing and Chemistry are all entity occurrences within the Subject entity.

Attribute

An entity is described by its attributes. Each attribute is a characteristic of the entity. For example, attributes of the Student entity would include studentID, firstname, surname and dateOfBirth.

Primary key

An attribute or combination of attributes that uniquely identifies one, and only one, entity occurrence is called a primary key. For example, the primary key of the Student entity would be studentID. A primary key is signified by underlining in the entity-relationship diagram.

Foreign key

An attribute in one table that uniquely identifies a row of another table. A foreign key is signified by an asterisk in the entity-relationship diagram.

Compound key

A compound key is a primary key that comprises two or more attributes. Each attribute that makes up a compound key is a primary key in its own right. For example, the primary key of the Class entity would be the compound key formed by combining subjectCode + teacherRef + columnID. In this example, subjectCode would be the primary key of the Subject entity, teacherRef would be the primary key of the Teacher entity and columnID would be the primary key of the Column entity.

Relationship

A relationship is a natural association between one or more entities. For example, Students learn Subjects and Teachers educate Students.

Cardinality

The cardinality of a relationship defines the number of participants in the relationship. It states the number of entity occurrences in one entity that are associated with one occurrence of the related entity. Cardinality can be:

- ◆ one-to-one
- ◆ one-to-many
- ◆ many-to-many

One-to-one relationship

In a one-to-one relationship, each entity occurrence in an entity is associated with one, and only one, entity occurrence within a related entity. For example, a School is managed by one, and only one, Headteacher, with a Headteacher managing one, and only one, School.

See *Entity-occurrence diagram example 1* and *Entity-relationship diagram example 1*

One-to-many relationship

In a one-to-many relationship, each entity occurrence in an entity can be associated with one or more entity occurrences in a related entity. For example, a School employs many Teachers and each of those Teachers is employed by one School.

See *Entity-occurrence diagram example 2* and *Entity-relationship diagram example 2*

Many-to-many relationship

In a many-to-many relationship, several entity occurrences in an entity can be associated with multiple entity occurrences in a related entity. For example, many Students study several different Subjects and each of those Subjects is studied by many Students. Direct many-to-many relationships between two entities cannot be implemented by a relational database system. To overcome this problem, many-to-many relationships can be resolved to form two one-to-many relationships.

See *Entity-occurrence diagram example 3* and *Entity-relationship diagram example 3*

Entity-occurrence diagrams

An entity-occurrence diagram illustrates the relationships between the entity occurrences of one entity, with the entity occurrences within a related entity. The creation of an entity-occurrence diagram helps to identify the cardinality of the relationship that exists between the two entities.

In an entity-occurrence diagram, each entity is shown as a tall oval. Inside each entity, each entity occurrence is represented by the value of its identifier and each relationship is illustrated by drawing a line between associated entity occurrences.

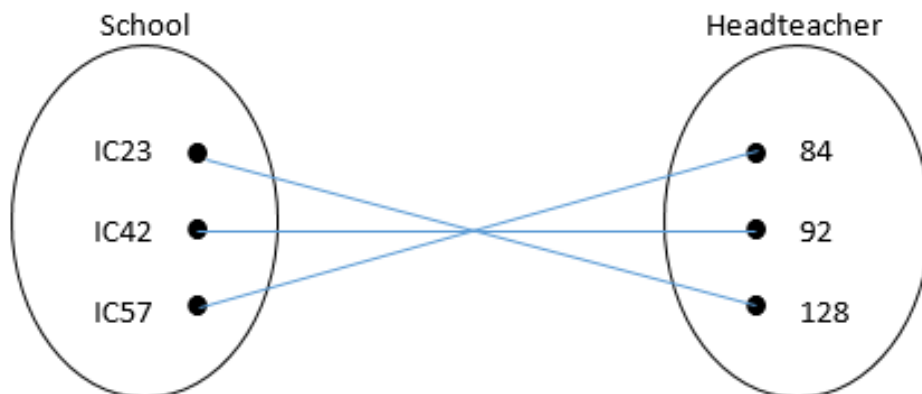
Examples of entity-occurrence diagrams are shown below.

Entity-occurrence diagram example 1: one-to-one relationship

The following table indicates which School is managed by which Headteacher and which Headteacher manages which School.

School	Headteacher
IC42	92
IC57	84
IC23	128

Here is the matching entity-occurrence diagram.



From this entity-occurrence diagram, we can see that each occurrence in the School entity has an association with one, and only one, entity occurrence in the Headteacher entity. Similarly, each occurrence in the Headteacher entity has an association with one, and only one, occurrence in the School entity.

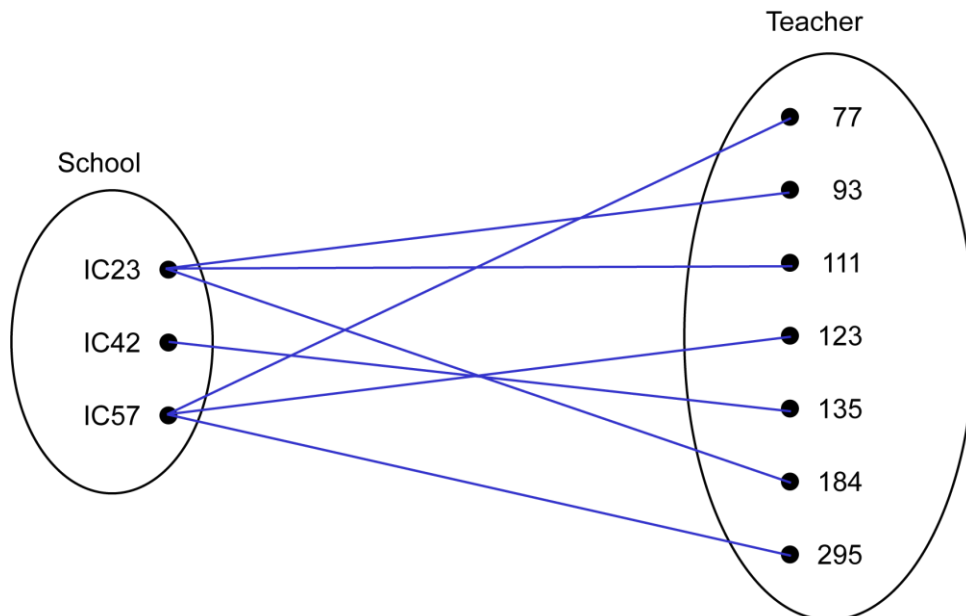
This confirms that there is a one-to-one relationship between the School and Headteacher entities.

Entity-occurrence diagram example 2: one-to-many relationship

The following table indicates which School employs which Teachers and which Teachers are employed by which School.

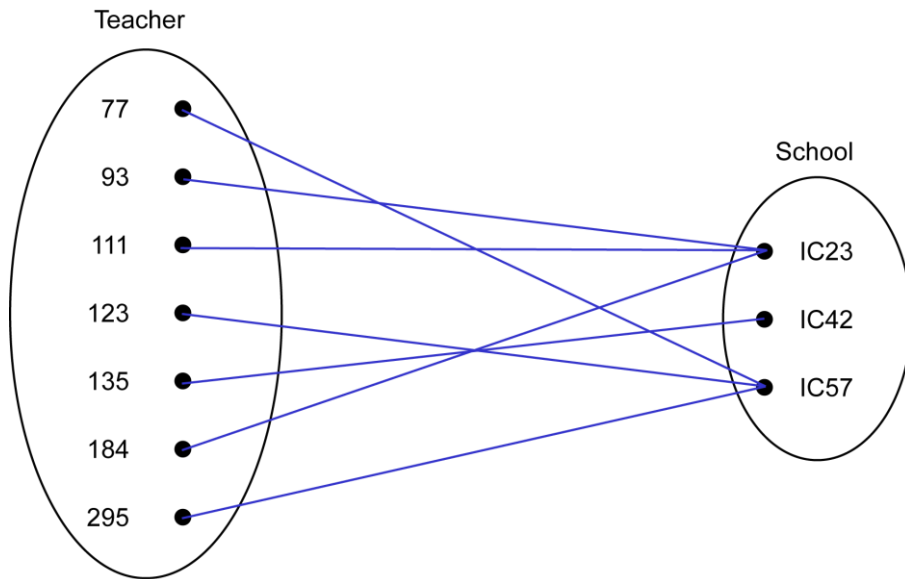
School	Teacher
IC42	135
IC57	123
IC23	111
IC23	184
IC57	77
IC57	295
IC23	93

Here is the matching entity-occurrence diagram.



From this entity-occurrence diagram, we can see that each occurrence in the School entity has an association with one or more entity occurrences in the Teacher entity. We can also see that each occurrence in the Teacher entity has an association with one, and only one, occurrence in the School entity. This confirms that there is a one-to-many relationship between the School and Teacher entities.

Note: this diagram is equivalent to the following entity-occurrence diagram.

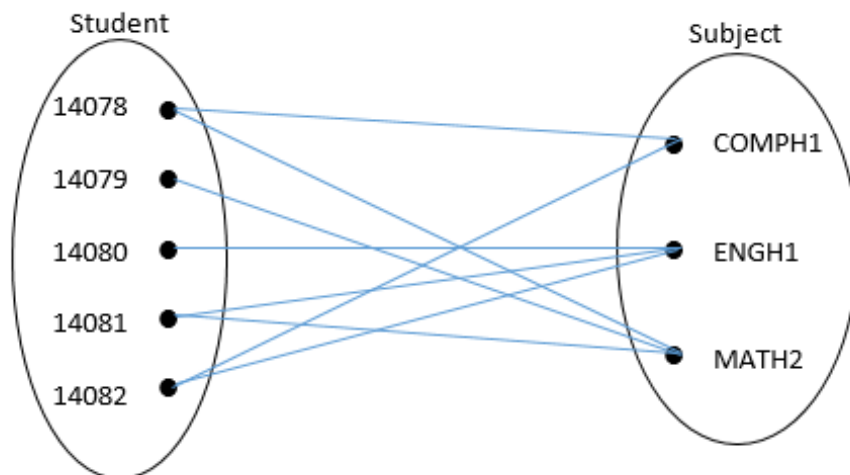


Entity-occurrence diagram example 3: many-to-many relationship

The following table indicates which Students study which Subjects and which Subjects are studied by which Students.

Student	Subject
14078	COMPH1
14079	MATH2
14080	ENGH1
14078	MATH2
14081	ENGH1
14082	ENGH1
14082	COMPH1
14081	MATH2

Here is the matching entity-occurrence diagram.

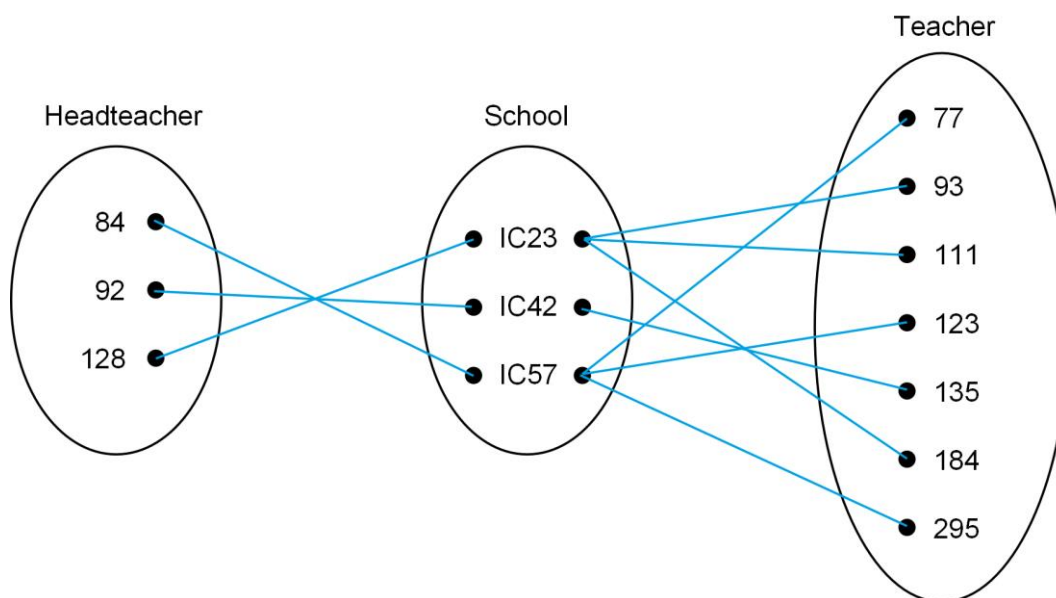


From this entity-occurrence diagram, we can see that each occurrence in the Student entity has an association with one or more entity occurrences in the Subject entity. We can also see that each occurrence in the Subject entity has an association with one or more occurrences in the Student entity.

This confirms that there is a many-to-many relationship between the Student and Subject entities.

Entity-occurrence diagram example 4: multiple entities

An entity-occurrence diagram may be used to illustrate the relationship between multiple entities as shown below:



Entity-relationship diagrams

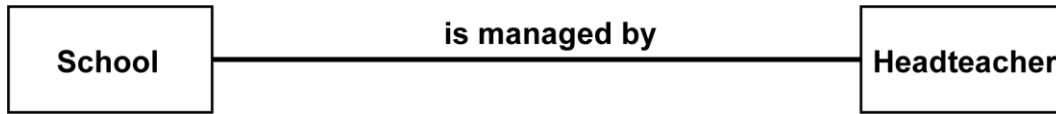
An entity-relationship diagram is a graphical representation of the entities in a system. It is used to summarise the relationship that exists between two or more entities. An entity-relationship diagram indicates:

- ◆ the name of each entity in the system
- ◆ the name of the relationship between two entities
- ◆ the cardinality of the relationship between two entities
- ◆ if required, the name of each attribute can be shown

The entities on an entity-relationship diagram are represented by **labelled rectangles**. If required, the attributes within each entity can be represented as **labelled ovals**. The relationship between two entities is represented by the **labelled line** which is used to join the entities. Although several different representations can be used, the entity-relationship diagrams in the examples below make use of the **crow's feet notation** to indicate the 'many' sides of a relationship.

Examples of entity-relationship diagrams are shown below.

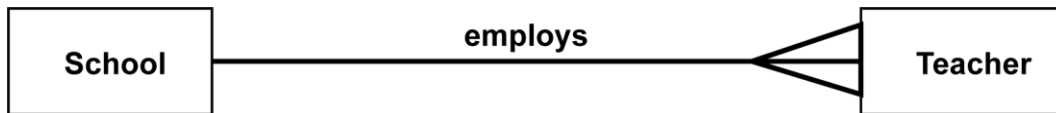
Entity-relationship diagram example 1: one-to-one relationship



This entity-relationship diagram illustrates the relationship between the two entities School and Headteacher. A description of the relationship can be generated by reading across the diagram in both directions:

Each School is managed by one, and exactly one, Headteacher while each Headteacher manages one, and only one, School.

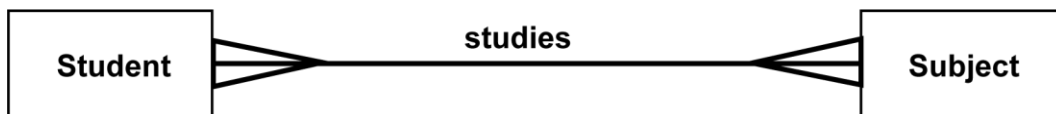
Entity-relationship diagram example 2: one-to-many relationship



This entity-relationship diagram illustrates the relationship between the two entities School and Teacher. A description of the relationship can be generated by reading across the diagram in both directions:

Each School employs one or more Teachers and each Teacher is employed by one, and only one, School.

Entity-relationship diagram example 3: many-to-many relationship



This entity-relationship diagram illustrates the relationship between the two entities Student and Subject. A description of the relationship can be generated by reading across the diagram in both directions:

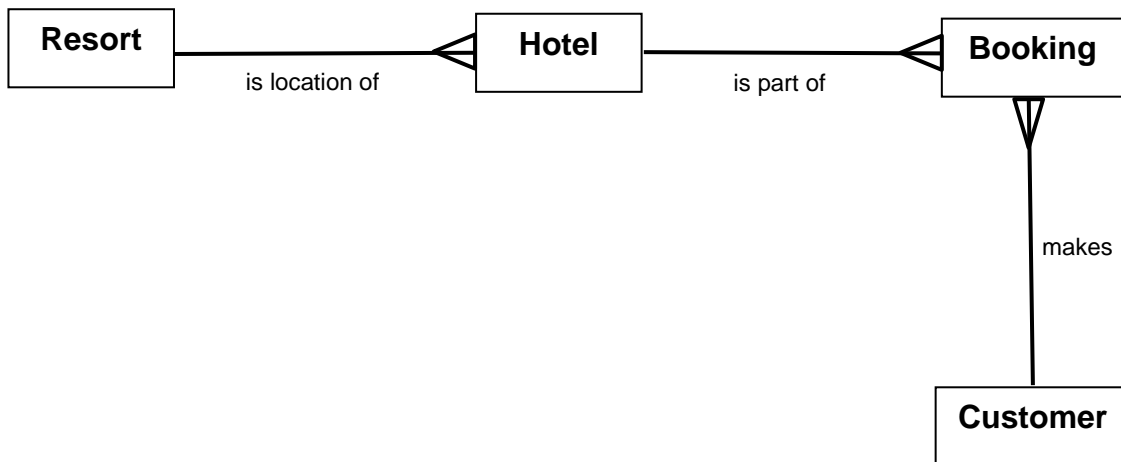
Each Student studies one or more Subjects and each Subject is studied by one or many Students.

Entity-relationship diagram example 4: worked example

A relational database is used by a travel agency to store details of Scottish holiday resorts and hotels in each resort. The database also stores details of customers and their hotel bookings. The Booking, Customer, Resort and Hotel details are arranged in four separate entities.

Entity: Resort	Entity: Hotel	Entity: Customer	Entity: Booking
<u>resortID</u> resortName resortType trainStation	<u>hotelRef</u> hotelName resortID* starRating seasonStartDate swimmingPool mealPlan checkInTime pricePersonNight	<u>customerNo</u> firstname surname address town postcode	<u>bookingNo</u> customerNo* hotelRef* startDate numberNights numberInParty

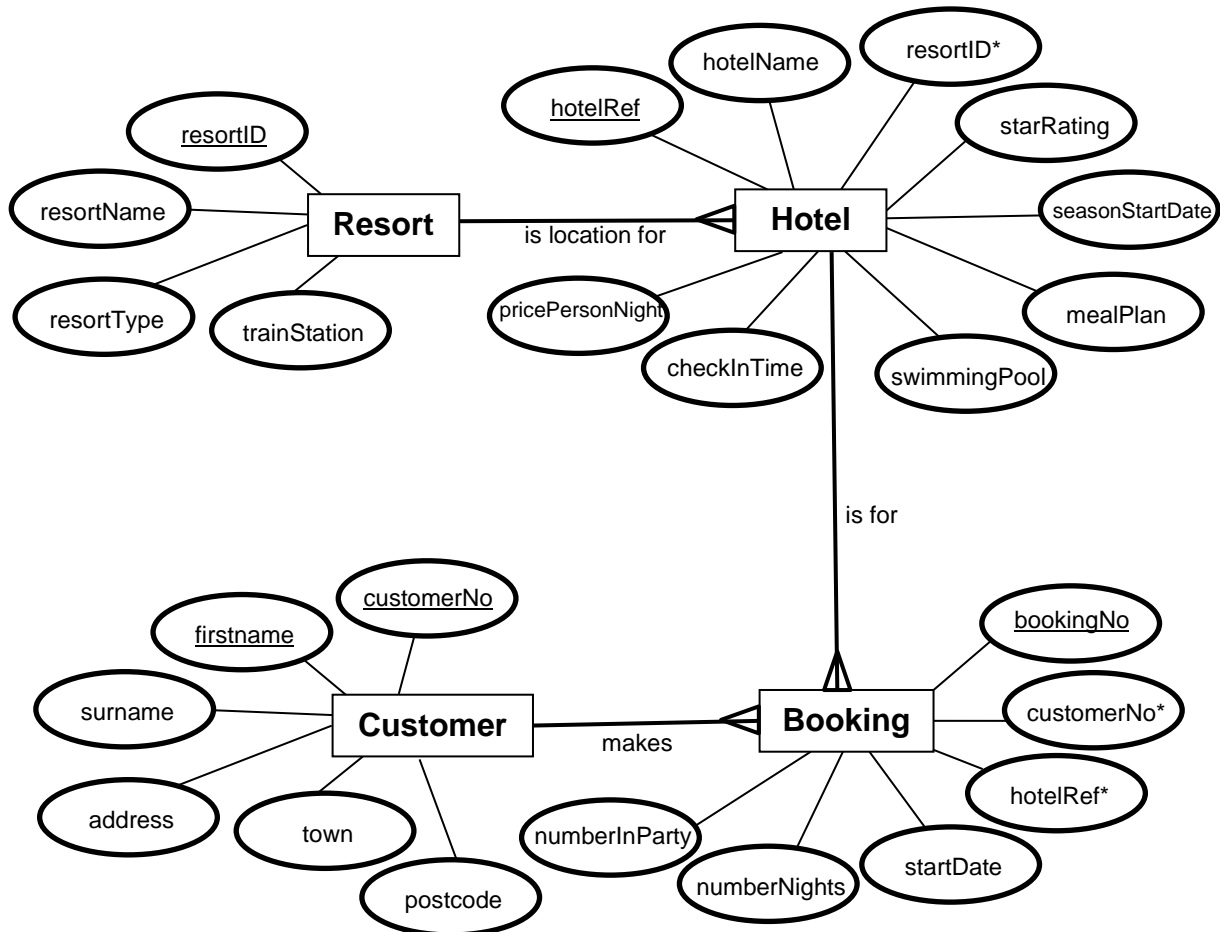
The matching entity-relationship diagram is shown below.



Entity-relationship diagram example 5: worked example

As exemplified in National 5, entity-relationship diagrams can also be used to illustrate the attributes that are stored in each entity.

Based on the details of the entities and attributes provided in *Entity-relationship diagram example 4*, the matching entity-relationship diagram is shown below:



Appendix 10: design of solution to database queries (DDD)

A travel agency uses a relational database to enable their employees to view details of hotels in Scottish holiday resorts and make bookings for customers. The details are stored in four separate tables called Hotel, Resort, Booking and Customer.

The structure of these tables is shown below:

Hotel	
	Field Name
🔑	hotelRef
	hotelName
	resortID
	starRating
	seasonStartDate
	swimmingPool
	mealPlan
	checkInTime
	pricePersonNight

Resort	
	Field Name
🔑	resortID
	resortName
	resortType
	trainStation

Booking	
	Field Name
🔑	bookingNo
	customer#
	hotelRef
	startDate
	numberNights
	numberInParty

Customer	
	Field Name
🔑	customer#
	firstname
	surname
	address
	town
	postcode

The design of the SQL query should indicate:

- ◆ any field(s) or computed values required
- ◆ the table(s) needed to provide all of the details required
- ◆ any search criteria to be applied
- ◆ what grouping is needed (if appropriate)
- ◆ the field(s) used to sort the data and the type(s) of sort required

Planning ahead helps to reduce the amount of frustration that candidates may otherwise encounter when working with the SQL code.

A simple table template, like in the examples below, can be used by candidates to indicate the planned design of the SQL query.

Example 1: design a query to display the name, swimming pool details, resort and resort type of any hotel in a coastal resort that starts with the letter 'A'.

Field(s) and calculation(s)	hotelName, swimmingPool, resortName, resortType
Table(s) and query	Hotel, Resort
Search criteria	resortType = "coastal" and resortName like "A%"
Grouping	
Sort order	

Example 2: design a query to display a customer's full name, booking number, start date, hotel name and resort name for all customers who have an 'h' as the second letter of their surname. List these details in alphabetical order of surname; listing customers with the same surname in order of the earliest holiday first.

Field(s) and calculation(s)	firstname, surname, bookingNo, startDate, hotelName, resortName
Table(s) and query	Customer, Booking, Hotel, Resort
Search criteria	surname LIKE "_h%"
Grouping	
Sort order	surname ASC, startDate ASC

Example 3: design a query that uses a readable heading to display the cheapest and dearest price per night.

Field(s) and calculation(s)	Dearest price per night = MAX(pricePersonNight), Cheapest price per night = MIN(pricePersonNight)
Table(s) and query	Hotel
Search criteria	
Grouping	
Sort order	

Example 4: design a query to display the average number of nights booked.

Field(s) and calculation(s)	AVG(numberNights)
Table(s) and query	Booking
Search criteria	
Grouping	
Sort order	

Example 5: design a query to display a list of the different types of resort, together with the number of resorts in each of those categories.

Field(s) and calculation(s)	resortType, COUNT(*)
Table(s) and query	Resort
Search criteria	
Grouping	resortType
Sort order	

Example 6: design a query to display the number of bookings for hotels in coastal resorts. Show the resort type and use a readable heading for the results returned by the aggregate function.

Field(s) and calculation(s)	resortType, Number of Hotels = COUNT(*)
Table(s) and query	Resort, Hotel, Booking
Search criteria	resortType = "coastal"
Grouping	resortType
Sort order	

Example 7: design a query to display a list of each type of meal plan, together with the number of bookings made for each of those meal plans. List the details from the least popular meal plan to the most popular.

Field(s) and calculation(s)	mealPlan, COUNT(*)
Table(s) and query	Hotel, Booking
Search criteria	
Grouping	mealPlan
Sort order	COUNT(*) ASC

Example 8: design a query that uses a readable heading to display the total number of people booked into a hotel in July.

Field(s) and calculation(s)	People booked in July = SUM(numberInParty)
Table(s) and query	Booking
Search criteria	startDate LIKE "%/07/%"
Grouping	
Sort order	

Example 9: design a SELECT query to display the hotel name and the improved rating, if all hotels in Ayr gain an extra star (use a readable heading to display the improved ratings).

Field(s) and calculation(s)	hotelName, Improved rating = starRating + 1
Table(s) and query	Hotel, Resort
Search criteria	resortName = "Ayr"
Grouping	
Sort order	

Example 10: design a query to display the surname, booking number, number of nights, number in party, price per night and the total cost of each booking (with a readable column heading). Display the dearest booking first.

Field(s) and calculation(s)	surname, bookingNo, numberNights, numberInParty, pricePersonNight, Total Cost = (numberNights * numberInParty * pricePersonNight)
Table(s) and query	Customer, Booking, Hotel
Search criteria	
Grouping	
Sort order	numberNights * numberInParty * pricePersonNight DESC

Example 11: design a query to display the name of the resorts that have hotels with the highest star rating together with the highest star rating (use a readable heading to display the highest star rating).

Since it is not possible to use an aggregate function in a WHERE clause, this solution requires **two** separate queries:

- ◆ the first query is a simple query to generate a single value — in this case, the highest star rating
- ◆ the second query makes use of the value generated by the first query

Query 1 — Find Maximum Rating

Field(s) and calculation(s)	Highest Star Rating = MAX(starRating)
Table(s) and query	Hotel
Search criteria	
Grouping	
Sort order	

Query 2 — Display Names Of Resorts with Maximum Rating

Field(s) and calculation(s)	resortName, [Highest Star Rating]
Table(s) and query	Hotel, [Find Maximum Rating]
Search criteria	starRating = Highest Star Rating
Grouping	
Sort order	

Example 12: design a query to display details of any hotel with a price per person that is below the average price per person. The query should display the hotel name, star rating, meal plan and the price per night. The dearest hotel should be listed first.

This solution requires **two** separate queries:

- ◆ the first query is used to generate a single value — in this case, the average price per night
- ◆ the second query makes use of the value generated by the first query

Query 1 — Find Average Per Person

Field(s) and calculation(s)	Average Price per Person (£) = AVG(pricePersonNight)
Table(s) and query	Hotel
Search criteria	
Grouping	
Sort order	

Query 2 — Display Details of Hotels with Price Per Person Below Average

Field(s) and calculation(s)	hotelName, starRating, mealPlan, pricePersonNight
Table(s) and query	Hotel, [Find Average per Person]
Search criteria	pricePersonNight = Average Price per Person (£)
Grouping	
Sort order	pricePersonNight DESC

Appendix 11: SQL (DDD)

Note: we are using the following relational database to exemplify the Higher SQL commands.

A travel agency uses a relational database to enable their employees to view details of hotels in Scottish holiday resorts and make bookings for customers. The details are stored in four separate tables called Hotel, Resort, Booking and Customer.

The structure of these tables is shown below:

Hotel		Resort		Booking	
Field Name		Field Name		Field Name	
hotelRef		resortID		bookingNo	
hotelName		resortName		customer#	
resortID		resortType		hotelRef	
starRating		trainStation		startDate	
seasonStartDate				numberNights	
swimmingPool				numberInParty	
mealPlan					
checkInTime					
pricePersonNight					
town					
postcode					

A sample record stored in each table is shown below:

Hotel table	
hotelRef	AY72
hotelName	Cliff Top
resortID	168
starRating	3
seasonStartDate	29/04/2018
swimmingPool	False
mealPlan	Half Board
checkInTime	14:30
pricePersonNight	58.99

Resort table	
resortID	168
resortName	Ayr
resortType	Coastal
trainStation	True

Booking table	
bookingNo	134
customer#	426
hotelRef	AY72
startDate	30/04/2018
numberNights	7
numberInParty	3

Customer table	
customer#	426
firstname	Omar
surname	Shaheed
address	26a High Bridge
town	Perth
postcode	PH12 34X

Note: The field customer# is enclosed in square brackets to avoid syntax errors.

Wildcards

A wildcard character is used to replace one or more characters in a string. Wildcards are useful in situations when incomplete information is available and it would be impossible to write a `WHERE` clause using one of the existing logical operators `=`, `<`, `>`, `≤` or `≥`.

Wildcard characters are used with the SQL `LIKE` operator. The `LIKE` operator is used in a `WHERE` clause to perform search operations. Two different wildcards can be used:

- `%` (the percent symbol) is used to represent zero, one or multiple characters
- `_` (the underscore symbol) is used to represent a single character

Note: MS Access uses:

- ◆ `*` rather than `%`
- ◆ `?` rather than `_`

The following are some examples of `LIKE` used with the wildcards:

Example	Purpose
<code>WHERE surname LIKE 'Thom%'</code>	Used to find any values in the surname field that start with "Thom"
<code>WHERE surname LIKE '%son'</code>	Used to find any values in the surname field that end with "son"
<code>WHERE surname LIKE '%is%'</code>	Used to find any values that have "is" anywhere in the surname field
<code>WHERE surname LIKE '_h%'</code>	Used to find any values in the surname field that have "h" as the second character
<code>WHERE surname LIKE 'm_ _ %'</code>	Used to find any values in the surname field that start with "m" and have at least 3 characters
<code>WHERE surname LIKE 'a%z'</code>	Used to find any values in the surname field that start with "a" and end with "z"

Example 1: used to search the database to display the name, swimming pool details, resort and resort type of any hotel in a coastal resort that starts with the letter 'A'.

```
SELECT hotelName, swimmingPool, resortName, resortType
FROM Hotel, Resort
WHERE Hotel.resortID = Resort.resortID AND resortName LIKE
'A%' AND resortType = 'coastal';
```

Example 2: used to display the customer's full name, booking number, start date, hotel name and resort name of all customers who has an 'h' as the second letter of their surname. These details should be listed in alphabetical order of surname; customers with the same surname should be listed so that the customer with the earliest holiday should be listed first.

```
SELECT firstname, surname, bookingNo, hotelName,
resortName, startDate

FROM Customer, Booking, Hotel, Resort

WHERE Customer.[customer#]=Booking.[customer#] AND
Booking.hotelRef=Hotel.hotelRef AND
Hotel.resortID=Resort.resortID AND surname LIKE '_h%'

ORDER BY surname ASC, startDate ASC;
```

Aggregate functions

Aggregate functions operate on a set of rows to return a single, statistical value. You apply an aggregate to a set of rows, which may be:

- ◆ all the rows in a table
- ◆ only those rows specified by a WHERE clause
- ◆ those rows created by a GROUP BY clause (see later)

The most common aggregate functions used are listed below:

Function	Description
AVG ()	returns the average value of a numeric column or expression
COUNT ()	returns the number of rows that match the criteria in the WHERE clause
MAX ()	returns the largest value of the selected column or expression
MIN ()	returns the smallest value of the selected column or expression
SUM ()	returns the total sum of a numeric column or expression

In the same way that pre-defined programming functions receive parameter values, SQL aggregate functions require an expression. This expression is usually a column name but it can be a column name together with an operator.

The following points should be noted:

- ◆ SUM () and AVG () can only be applied to numeric data types; MIN () and MAX () work with characters, numeric, and date/time datatypes; COUNT () works with all data types.
- ◆ All aggregate functions except, COUNT (), ignore nulls.

- ◆ `COUNT()` always returns a positive integer or zero. The other aggregate functions return null if the set contains no rows or contains rows with only nulls.
- ◆ An aggregate expression cannot be used in a `WHERE` clause.
- ◆ It is possible to use more than one aggregate expression in a `SELECT` statement as shown here:

```
SELECT MIN(price), MAX(price)
FROM Product;
```

- ◆ Mixing non-aggregate and aggregate expressions in a `SELECT` statement is not permitted. A `SELECT` statement must contain either all non-aggregate expressions or all aggregate expressions. The query below is illegal, as it mixes non-aggregate `productName` with the aggregate function `MAX`.

```
SELECT productName, MAX(price)
FROM Product;
```

Example 3: uses readable headings to display the cheapest and dearest price per night.

```
SELECT MIN(pricePersonNight) AS [Cheapest Price per
Night], MAX(pricePersonNight) AS [Dearest Price perNight]
FROM Hotel;
```

Example 4: used to display the average number of nights booked.

```
SELECT ROUND(AVG(numberNights),2)
FROM Booking;
```

Note: the SQL `ROUND()` function is used to round the average to 2 decimal places.

Example 5: used to display a list of the different types of resort together with the number of resorts in each of those categories.

```
SELECT resortType, COUNT(*)
FROM Resort
GROUP BY resortType;
```

Example 6: uses a readable heading to display the total number of people booked into a hotel in July.

```
SELECT SUM(numberInParty) AS [People on holiday in July]
FROM Booking
WHERE startDate LIKE '%/07/%';
```

Computed values with aliases

Arithmetic expressions can be used to compute values as part of a `SELECT` query. The arithmetic expressions can contain column names, numeric numbers, and arithmetic operators.

Whenever a value is generated by a query, it is allocated its own column in the query answer table. A computed value is temporary — it only exists within the query. Because of this, computed values are not stored in the database, which eliminates the need to store data that can be computed at run-time.

An **alias** can be used to give any column in an answer table a temporary name. Doing this makes the headings in the answer table more readable. Since it is generated at run-time, an alias only exists for the duration of the query. An alias is listed in the `SELECT` list by using the `AS` statement.

For example, the query below will display the name, price, quantity and cost of each product in a specified order:

```
SELECT productName AS ['Product Name'], price, quantity,
price*quantity
FROM Product, Order
WHERE Product.productID = [Order].productID AND order# =
123456;
```

Executing the query produces the answer table below:

Product Name	price	quantity	price*quantity
Oven cleaner	3.45	3	10.35
Carpet cleaner	4.16	2	8.32
Bleach	1.99	5	9.95

We can make the answer table more readable by using an alias:

```
SELECT productName AS ['Product Name'], price, quantity,
price*quantity AS ['Product Cost']
FROM Product, Order
WHERE Product.productID = [Order].productID AND order# =
123456;
```

Executing the updated query produces the answer table below:

Product Name	price	quantity	Product Cost
Oven cleaner	3.45	3	10.35
Carpet cleaner	4.16	2	8.32
Bleach	1.99	5	9.95

The column headings in this second answer table are more readable than those in the first answer table, due to the use of aliases in the second query.

GROUP BY

The `GROUP BY` clause is used in a `SELECT` to form sets (or groups) of records. It does this by gathering together all records that have identical data in the specified column(s).

When used with an aggregate function, `GROUP BY` ensures that one result is returned for each set of grouped records. This makes it possible to mix non-aggregate and aggregate expressions for grouping columns; without `GROUP BY`, this is not possible.

For example, the query shown below is used to display a list of product categories together with the dearest product in each of those categories. The category with the cheapest product is listed first.

```
SELECT productCategory, MAX(price)
FROM Product
GROUP BY productCategory
ORDER BY MAX(price) ASC;
```

Note: whenever a single query has both `GROUP BY` and `ORDER BY` clauses, the `GROUP BY` clause **always** precedes the `ORDER BY` clause. If the clauses are reversed, a syntax error will be generated.

Example 7: use to display the number of bookings for hotels in coastal resorts. Show the resort type and use a readable heading for the results returned by the aggregate function.

```
SELECT resortType, COUNT(*) AS [Number of Bookings]
FROM Resort, Hotel, Booking
WHERE Resort.resortID = Hotel.resortID AND Hotel.hotelRef
= Booking.hotelRef AND resortType = 'coastal'
GROUP BY resortType;
```

Example 8: use to display a list of each type of meal plan together with the number of bookings made for each of those meal plans. The details should be listed from least popular meal plan to most popular.

```
SELECT mealPlan, COUNT(*)
FROM Hotel, Booking
WHERE Hotel.hotelRef = Booking.hotelRef
GROUP BY mealPlan
ORDER BY COUNT(*) ASC;
```

Note: the sequencing of the `GROUP BY` and `ORDER BY` clauses.

UPDATE query used to edit more than one field

The `UPDATE` query is used at National 5 to edit the value(s) stored in one or more fields using a single criteria. For example:

```
UPDATE Product
SET productName = "Oven cleaner exceptional", price =
price * 1.10
WHERE productName = "Oven cleaner";
```

This query will update the product name and the price of the product called 'Oven cleaner'.

At Higher level, candidates are required to implement complexity in the `WHERE` clause. This could include multiple conditions, the use of alias, wildcard or conditions with calculated values. The general syntax of this `UPDATE` query is:

```
UPDATE tableName
SET field1 TO expression, field2 TO expression, ... ..
WHERE criteria to be matched;
```

Note: each expression used in the `SET` clause can be a specific value or an expression (which can, if required, use arithmetic operators).

Example 9: customer Omar Shaheed has moved. Use an `UPDATE` query to edit his details (his new address is provided below):

New Address	31 Pike Place
New Postcode	PH31 31P

```
UPDATE Customer
SET address = '31 Pike Place', postcode = 'PH31 31P'
WHERE firstname = 'Omar' AND surname = 'Shaheed';
```

Example 10: all the hotels in Fort William (hotel references starting with the letters FW) have gained an extra star and have increased the price per night by 4%. Edit the relevant details in the database.

```
UPDATE Hotel
SET pricePersonNight = ROUND(pricePerNight * 1.04,2),
starRating = starRating + 1
WHERE hotelRef LIKE 'FW%';
```

Using the result of a query

The result of stored query can be used within another query. The stored answer table is considered as a table by the database engine and any value generated by the query can be used as part of the `WHERE` clause in a second query. This is especially useful when working with aggregate functions.

Example 11: used to display the name of the resorts that have hotels with the highest star rating together with the highest star rating (use a readable heading to display the highest star rating).

Query 1 — Find Maximum Rating

```
SELECT MAX(starRating) AS [Highest Star Rating]
FROM Hotel;
```

Query 2 — Display Names Of Resorts with Maximum Rating

```
SELECT resortName AS [Resorts with Highest Rated Hotels],
[Highest Rating]
FROM Resort, Hotel, [Find Maximum Rating]
WHERE Resort.resortID = Hotel.resortID
AND starRating = [Highest Star Rating];
```

A subclause may be used to combine the two queries. Using subclauses is beyond the scope of the Higher course and will not be assessed. However, it may be useful to help to understand how the result of a query can be used within another query. The SQL statement below demonstrates using an aggregate function as part of a condition, allowing the selection of all records that match the maximum rating.

```
SELECT resortName AS [Resorts with highest rating]
FROM Resort,Hotel
WHERE Resort.resortID = Hotel.resortID
AND starRating = (SELECT MAX(starRating) FROM Hotel);;
```

Example 12: used to display details of any hotel with a price per person that is below the average price per person. The query should display the hotel name, star rating, meal plan and the price per night. The dearest hotel should be listed first.

Query 1 — Find Average Per Person

```
SELECT Round(AVG(pricePersonNight),2) AS [Average Price
per Person (£)]
FROM Hotel;
```

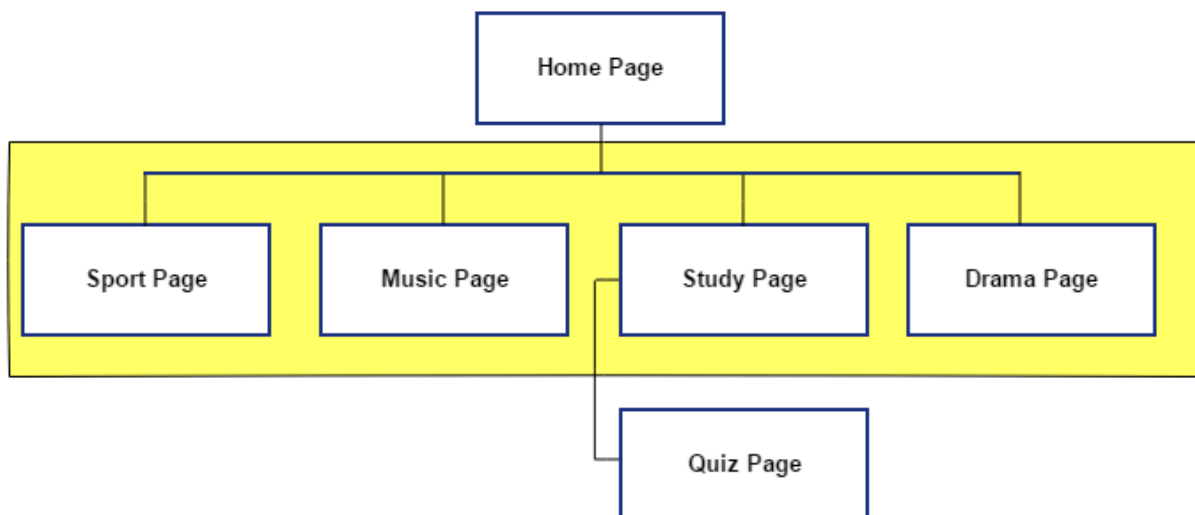
Query 2 — Display Details of Hotels with Price Per Person Below Average

```
SELECT hotelName, starRating, mealPlan, [Average Price per
Person (£)]
FROM Hotel, [Find Average Per Person]
WHERE pricePersonNight < [Average Price per Person (£)]
ORDER BY pricePersonNight DESC;
```

Appendix 12: design (WDD)

The design of the structure of a website is vital to ensure that users can easily find information on what could be thousands of individual pages. Hierarchical structures, using navigational bars, allow navigation through multi-level websites to be logical and straightforward.

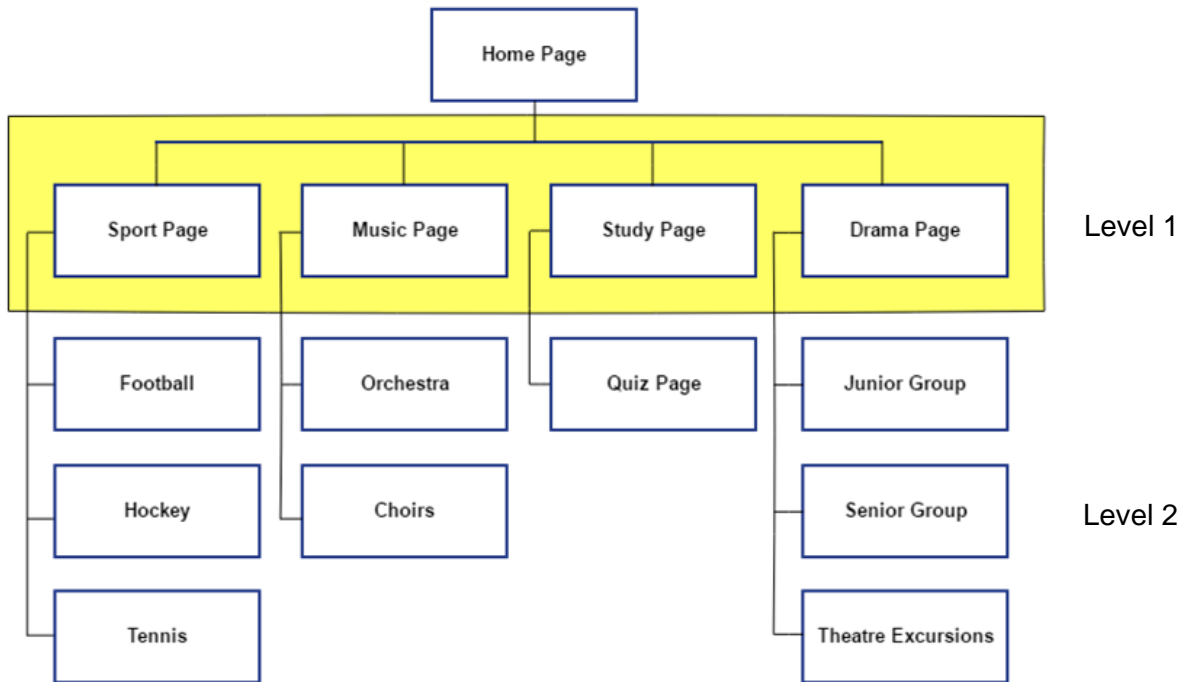
There are many design notations used to demonstrate the navigational structure. However, SQA will always use a shaded background to show the pages that are part of the horizontal navigational bar, which are at the top of each page in the site. The home page naturally is included in the navigational bar. The horizontal lines show the multi-links that link them all together. The vertical lines show the links that are only one way, from top to bottom.



Example

Penny High School is developing a new website. The website will have a multi-level structure, consisting of a home page with a horizontal navigation bar that gives clear links to four main areas (pages). Each of the four main pages will have links to relevant sub-pages.

The following diagram shows the navigational structure of the Penny High School website:



The user-interface planning should 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 horizontal and vertical position of:

- ◆ navigational bars
- ◆ 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
- ◆ any form inputs

together with intended position and type of all hyperlinks on the page.

Penny High Activities - Menu Page

School Activities text heading

Football.jpeg

[Home link](#) [Sport link](#) [Music link](#) [Study link](#) [Drama link](#)

Intro Text

Newslogo.jpeg

Daily Message Text

Hockey.jpeg Guitar.jpeg Drama.jpeg Pupils.jpeg Rugby.jpeg Band.jpeg

Contact Details Text

Penny High Activities - Sport Page

School Activities text heading

Football.jpeg

[Home link](#) [Sport link](#) [Music link](#) [Study link](#) [Drama link](#)

Sport Intro Text

Newslogo.jpeg

Daily Message Text

Footteam.jpeg TableTennis.jpeg Judo.jpeg

Contact Details Text

Penny High Activities - Music Page

School Activities text heading

Football.jpeg

[Home link](#) [Sport link](#) [Music link](#) [Study link](#) [Drama link](#)

Music Intro Text

Newslogo.jpeg Latest Message Text

Instructions for reveal boxes

Reveal Junior Choir Reveal Senior Choir Reveal Orchestra

Contact Details Text

Penny High Activities - Study Page

School Activities text heading

Football.jpeg

[Home link](#) [Sport link](#) [Music link](#) [Study link](#) [Drama link](#)

Study Intro Text

Newslogo.jpeg Latest Message Text

Study Technique text [Quiz](#)

Visual.jpeg Tactile.jpeg Audio.jpeg

text revealed by putting mouse over graphic above

Contact Details Text

Penny High Activities - Quiz Page

School Activities text heading

Football.jpeg

[Home link](#) [Sport link](#) [Music link](#) [Study link](#) [Drama link](#)

Quiz questions

Reveal

Quiz questions

Reveal

Quiz questions

Reveal

Contact Details Text

Penny High Activities - Drama Page

School Activities text heading

Football.jpeg

[Home link](#) [Sport link](#) [Music link](#) [Study link](#) [Drama link](#)

Drama info text

Book.jpeg

First Name text input box - No validation

Last Name text input box - No validation

Drop down menu to pick play ▼

Number of tickets required - range from 1 - 3

Select Age Radio buttons

12 - 14 15 - 16 over 16

Newslogo.jpeg

Latest Message Text

Special Requirements text input box - No validation

Contact Details Text

submit

Appendix 13: Cascading Style Sheets (CSS) — controlling appearance and positioning (WDD)

The following CSS declarations control the appearance and position of HTML page elements:

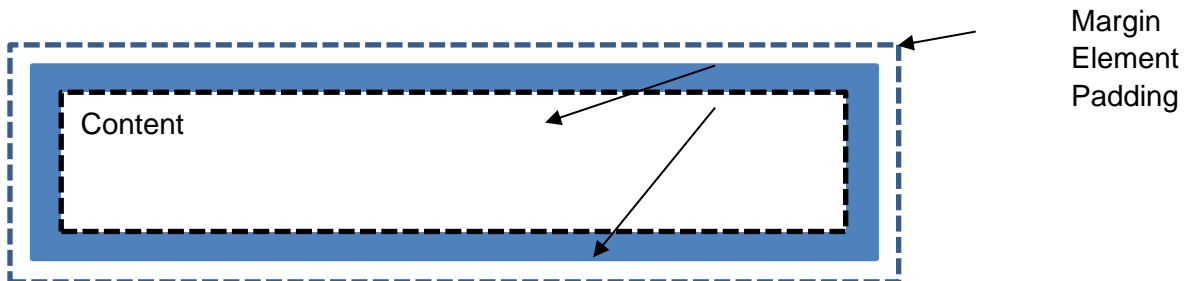
- ◆ display — block, inline, none
- ◆ float — left, right
- ◆ clear — both
- ◆ margins/padding
- ◆ sizes — height, width

The following examples have been taken from the *Higher Computing Science example website*. The example website can be downloaded as a zip file from the [Higher Computing Science](#) page on SQA's website. To view the full code in context, open and view the source code from the pages noted.

The box model

Margins and padding are used to push content away from the outer and inner edge of elements.

The box model allows us to define the space between elements.



Margin: declares a transparent area around the outside of an element. This pushes the element away from other adjacent elements.

Padding: declares a transparent area inside the edge of the element. This pushes content in from the edge of the element.

Universal selector

Browsers use default settings for margins and padding when displaying HTML elements. To override these defaults, a universal selector (*) can be used at the top of a stylesheet to set the margin and padding of every element to 0.

```
* {margin:0;padding:0}
```

Using the universal selector this way allows candidates to witness only the margins and paddings they actually code.

Note: universal selectors are not included in Higher content but can be useful when teaching.

Margins (all pages): main page areas

Margins can be declared as:

- ◆ margin
- ◆ margin-top
- ◆ margin-bottom
- ◆ margin-left
- ◆ margin-right

These properties may have the values:

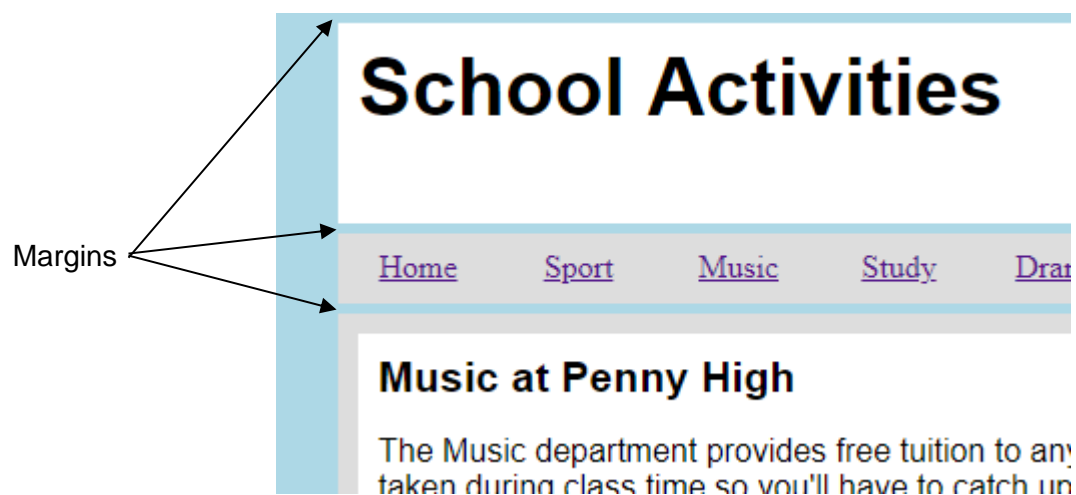
- ◆ auto (calculated by the browser)
- ◆ length (usually in pixels)

Margins can be declared on all four sides of an element simultaneously using the abbreviation: `margin:10px` .

In the example website, CSS declarations for the main page elements (<header>, <nav>, <div> and <footer>) are used to separate out content when displayed.

The small gap at the top of each area is implemented by styling a margin of 5 pixels at the top of the four elements. For example:

```
header, nav, main, footer {margin-top:5px}
```



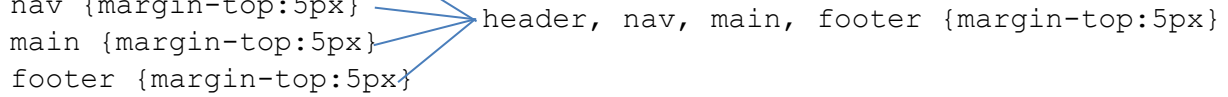
Grouping selectors using commas reduces the amount of code required and is more efficient than writing each CSS declaration separately.

Separate declarations

```
header {margin-top:5px}
nav {margin-top:5px}
main {margin-top:5px}
footer {margin-top:5px}
```

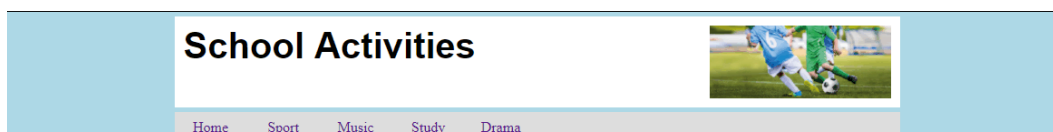
Grouped declaration

```
header, nav, main, footer {margin-top:5px}
```



Auto margins are used to position an element in the middle of the browser's window or within another element. In the example website, the 800 pixel wide page <body> element is always positioned in the middle of the window:

```
body{margin:auto}
```



Padding (drama page): sub-division of content

Padding can be declared as:

- ◆ padding
- ◆ padding-top
- ◆ padding-bottom
- ◆ padding-left
- ◆ padding-right

These properties may have the value:

- ◆ length (usually in pixels)


Padding can be declared on all four sides of an element simultaneously using the abbreviation: `padding:10px`

White space around page content, aids the usability of a web page. Appropriate use of white space can also aid readability of text.

Using padding to move content in from the edge of an element is one method to generate white space.

With padding: header, div, main, footer, section {padding: 10px}


School Activities



[Home](#) [Sport](#) [Music](#) [Study](#) [Drama](#)

Drama Opportunities

The drama department offer a variety of after school activities covering different acting experiences. These include groups on Shakespeare, improvisation and play writing. These clubs are used to prepare participants for the annual school show which is presented in collaboration with the Music department. There are also several annual visits to local performances of plays and musicals.



If you wish to put your name down for a Drama department visit to a performance please fill in the form on the right:

First name:


Last name:

Select play:

Number of Tickets Required (between 1 and 3):

Choose your age:
 12-14 15,16 over 16

If required, please delete and state any special requirements:




Thursday 13th June.
The Drama Department are organising a trip to see "The Curious Incident of the Dog in the Night-Time". Please complete the form above to enter your name for this outing or other upcoming events.

Contact Us

Landline	Mobile	E-mail
01314449999	0797575364	penny.high@midlands.gov.uk

Without padding: header, div, main, footer, section {padding: 0px}


School Activities



[Home](#) [Sport](#) [Music](#) [Study](#) [Drama](#)

Drama Opportunities

The drama department offer a variety of after school activities covering different acting experiences. These include groups on Shakespeare, improvisation and play writing. These clubs are used to prepare participants for the annual school show which is presented in collaboration with the Music department. There are also several annual visits to local performances of plays and musicals.



If you wish to put your name down for a Drama department visit to a performance please fill in the form on the right:

First name:


Last name:

Select play:

Number of Tickets Required (between 1 and 3):

Choose your age:
 12-14 15,16 over 16

If required, please delete and state any special requirements:



Thursday 13th June.
The Drama Department are organising a trip to see "The Curious Incident of the Dog in the Night-Time". Please complete the form above to enter your name for this outing or other upcoming events.

Contact Us

Landline	Mobile	E-mail
01314449999	0797575364	penny.high@midlands.gov.uk

Sizes — height/width (all pages)

Web page design and implementation involves creating areas that either have a fixed size or change size with content or display (changing window size or resolution).

In the example website, the <body> of each web page is set to a fixed width of 800px:

```
body{width:800px}
```

The height of the <header>, <nav> and <footer> elements are all set to a fixed size, remaining constant throughout the website:

```
header {height:80px}  
footer {height:60px}  
nav {height:35px}
```

The <main> element, which holds content of the pages, is not declared as a fixed size, as it changes according to the differing amount of content on each page.

Float — left/right (all pages)

An element can be positioned on the left or right of its container, using the float property.

In the example website, the element has been positioned on the right of its container, the <header> element:

HTML

```
<header>  
<h1>School Activities</h1>  
  
</header>
```

CSS

```
.imageBanner {width:200px;height:80px;float:right}
```

School Activities



Float can also be used to word-wrap text around an image, as demonstrated in the drama page below. The addition of margins creates white space between the graphic and the text.

Drama Opportunities

The drama department offer a variety of after school activities covering different acting experiences. These include groups on Shakespeare, improvisation and play writing. These clubs are used to prepare participants for the annual school show which is presented in collaboration with the Music department. There are also several annual visits to local performances of plays and musicals.

If you wish to put your name down for a Drama department visit to a performance please fill in the form on the right:



HTML

```
<p> 
```

```
The drama department offer a variety of after school
activities covering different acting experiences;
Shakespeare, improvisation, play writing. These clubs are
used to prepare participants for the annual school show
which is presented in collaboration with the Music
department. There are also several annual visits to local
performances.</p><p>If you wish to put your name down for a
Drama department visit to a performance please fill in the
form:</p>
```

CSS

```
.imageIconRight {width:60px;height:90px;float:right;margin-
left:10px;margin-bottom:10px}
```

Clear (all pages):

The effect of floating elements continues until cancelled, using the clear property on a subsequent element.

To ensure that the four main page elements <header>, <nav>, <main> and <footer> start a new line and remain unaffected by any float properties applied elsewhere in the page, clear:both was implemented for these elements.

```
header, nav, main, footer {display:block;clear:both}
```

Display — block/inline/none (all pages):

HTML elements have default display values depending on the type of element. This value specifies how or if an element is to be displayed. The default display value for most elements is block or inline:

`display:block` — an element takes up the entire width of its container

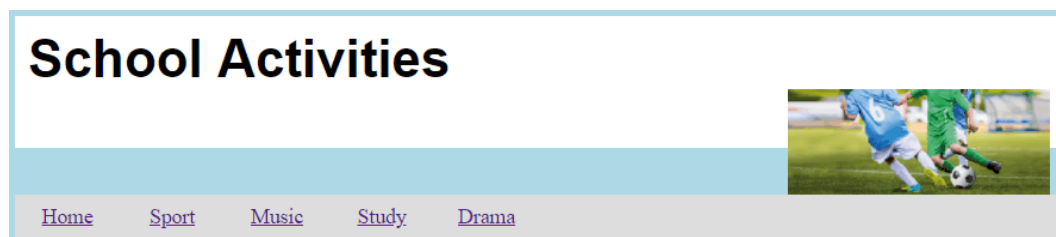
`display:inline` — an element takes up only as much room as necessary

`display:none` — the element is not visible. The space where the element should be collapses as if there was no content in that place.

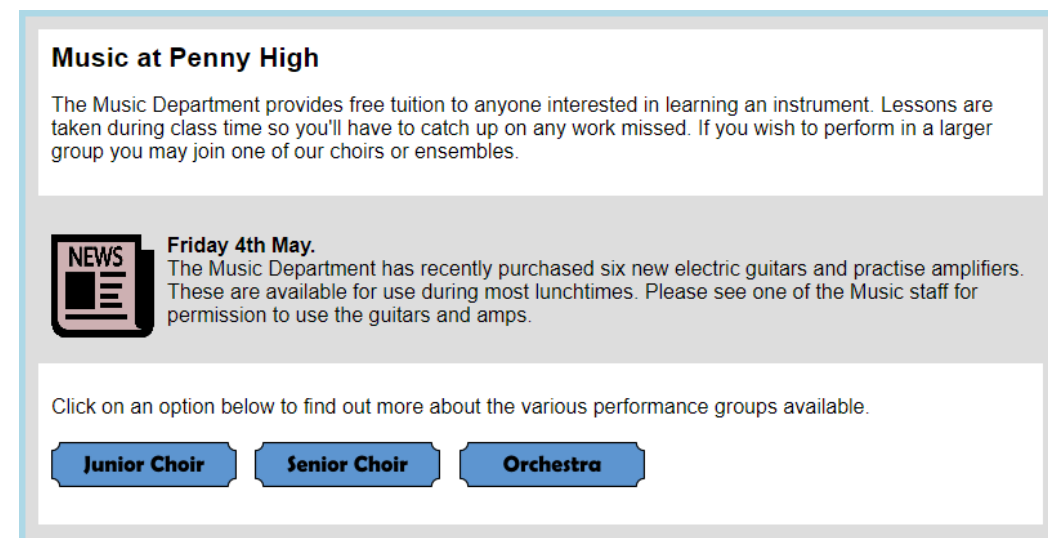
In the <header> element of the example website, the <h1> heading uses `display:inline`. This allows the image to be floated level with the heading.



If the <h1> element used `display:block` it would force the image to float on a new line.



In the music page, `display:none` is used to hide three <section> elements when the page loads.

A screenshot of a music department page. The page has a light blue border. At the top, the heading 'Music at Penny High' is displayed in a bold, black font. Below the heading is a paragraph of text: 'The Music Department provides free tuition to anyone interested in learning an instrument. Lessons are taken during class time so you'll have to catch up on any work missed. If you wish to perform in a larger group you may join one of our choirs or ensembles.' Below this text is a 'NEWS' section. It features a small icon of a newspaper with the word 'NEWS' written on it. To the right of the icon is the text: 'Friday 4th May. The Music Department has recently purchased six new electric guitars and practise amplifiers. These are available for use during most lunchtimes. Please see one of the Music staff for permission to use the guitars and amps.' At the bottom of the page, there is a paragraph of text: 'Click on an option below to find out more about the various performance groups available.' Below this text are three blue buttons with white text: 'Junior Choir', 'Senior Choir', and 'Orchestra'.

HTML


```
<section id="junior" style="display:none;height:100px">

<p><b>Junior Choir</b><br>The junior meet every Monday
lunchtime. It is open to anyone from S1 to S3.</p>
</section>
```

The above section is revealed when the 'Junior Choir' image on the page is clicked. This is achieved using a JavaScript onclick event to change the display property of the <section> to block.


Music at Penny High

The Music Department provides free tuition to anyone interested in learning an instrument. Lessons are taken during class time so you'll have to catch up on any work missed. If you wish to perform in a larger group you may join one of our choirs or ensembles.

 **Friday 4th May.**
The Music Department has recently purchased six new electric guitars and practise amplifiers. These are available for use during most lunchtimes. Please see one of the Music staff for permission to use the guitars and amps.

Click on an option below to find out more about the various performance groups available.

[Junior Choir](#) [Senior Choir](#) [Orchestra](#)

 **Junior Choir**
The junior meet every Monday lunchtime. It is open to anyone from S1 to S3.

For more details, see appendix 17: JavaScript.

Appendix 14: Cascading Style Sheets (CSS) — horizontal navigation bar (WDD)

The HTML 5 <nav> element is used to contain website navigation links, usually as a navigation bar. A navigation bar is declared as an unordered list of hyperlinks:

```
<nav>
  <ul>
    ...
    <li>hyperlink</li>
    <li>hyperlink</li>
    ...
  </ul>
</nav>
```

The look and feel of a navigation bar is then created using CSS declarations.

List of hyperlinks before and after CSS declarations

Before CSS declarations are added, the list of HTML hyperlinks are displayed as a simple bullet point list:

- 
- [Home](#)
 - [Sport](#)
 - [Music](#)
 - [Study](#)
 - [Drama](#)

HTML

```
<nav>
  <ul>
    <li><a href="home.html">Home</a></li>
    <li><a href="sport.html">Sport</a></li>
    <li><a href="music.html">Music</a></li>
    <li><a href="study.html">Study</a></li>
    <li><a href="drama.html">Drama</a></li>
  </ul>
</nav>
```

The addition of CSS declarations alter the position, look and behaviour of the list to create a navigation bar.



[Home](#) [Sport](#) [Music](#) **[Study](#)** [Drama](#)

CSS

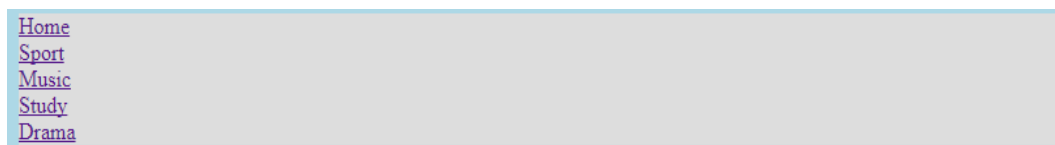
```
nav ul {list-style-type:none;}
nav ul li {float:left;width:80px;text-align:center}
nav ul li a {display:block;padding:8px}
nav ul li a:hover {background-color:#000;color:White}
```

The following examples breakdown the above code and provide explanations of each declaration. They have been taken from the navigation bar used within the *Higher Computing Science example website*. The example website can be downloaded as a zip file from the [Higher Computing Science](#) page on SQA's website. To view the full code in context, open and view the source code from any of the example web pages along with the external CSS file.

Example: removing bullets points

Declaring the `list-style-type` of the unordered list element as `none` removes the bullet points from the list:

```
nav ul {list-style-type:none}
```

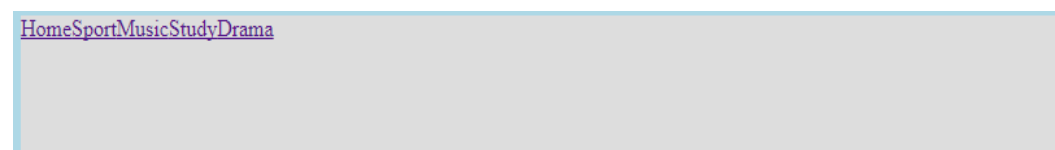


Using descendant selectors (`nav ul`) ensures that the style is only applied to the `ul` element within the `nav` element.

Example: positioning list items horizontally

To ensure the list of hyperlinks is distributed horizontally, each `` element is floated to the left:

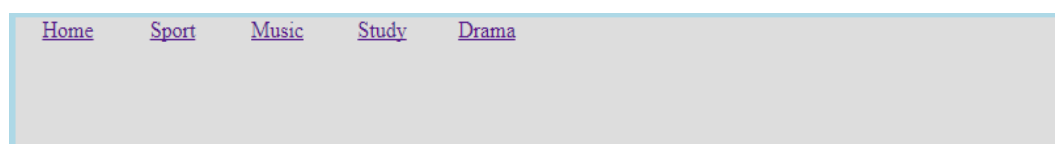
```
nav ul li {float:left}
```



Example: spacing out each list item

To create space between each list item, a width is declared and the link text is centred within each width:

```
nav ul li {float:left;width:80px;text-align:center}
```



Example: creating clickable boxes

In most navigation bars, clicking the area around the hyperlink also selects the link. A clickable box area around the link text is achieved by displaying the <a> element as a block:

```
nav ul li a {display:block}
```

Example: controlling vertical alignment

The vertical positioning of the link text in the navigation bar is controlled by declaring the height of the <nav> element and including padding within the <a> element:

```
nav {height:35px}  
nav ul li a {display:block;padding:8px}
```



Example: adding interactive colours

The state of the <a> element can be styled to change the background colour and text colour of each link, when the mouse hovers over an <a> element:

```
nav ul li a:hover {background-color:#000;color:White}
```

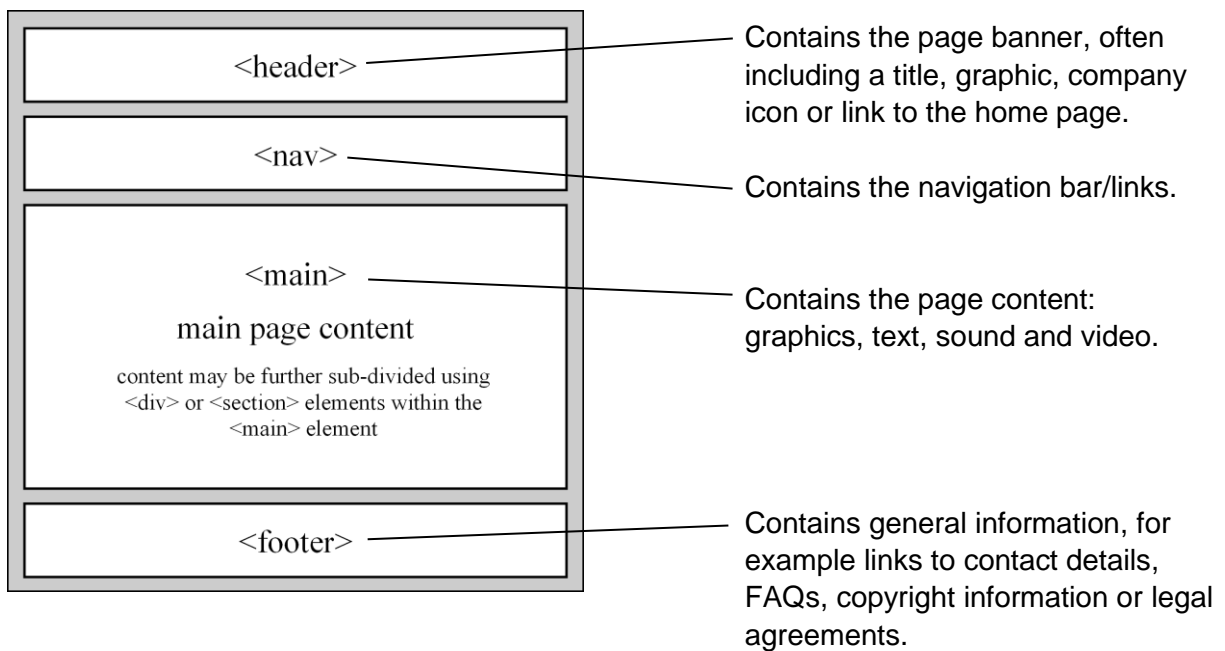


Appendix 15: HTML — page layout (WDD)

HTML 5 introduces new elements used to define different areas of a web page. Elements implemented at Higher level are:

- ◆ <header>
- ◆ <nav>
- ◆ <section>
- ◆ <footer>

These are implemented as shown below:



The following examples have been taken from the *Higher Computing Science example website*. The example website can be downloaded as a zip file from the [Higher Computing Science](#) page on SQA's website. To view the full code in context, open and view the source code from the pages noted in each of the examples.

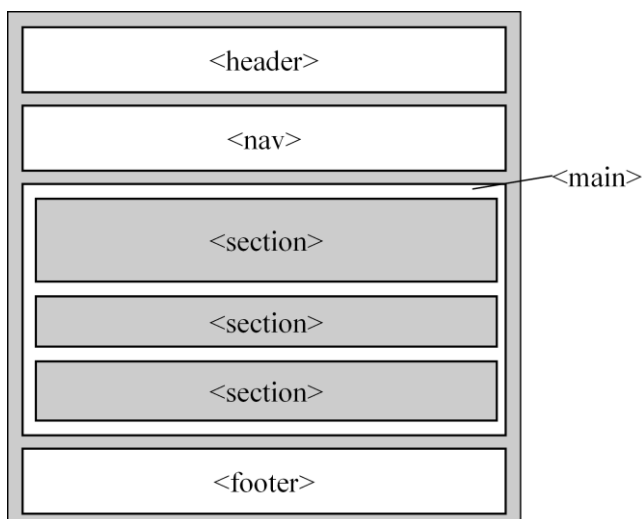
Simple page layout (study quiz page): example 1

The example below implements a simple page, with a main element used to contain the page content. The content is then sub-divided using section elements:

The screenshot shows a web page titled "School Activities" with a navigation menu (Home, Sport, Music, Study, Drama) and a quiz section titled "How healthy are your study practices?". The quiz contains three questions, each with a "Reveal" button. A footer section titled "Contact Us" provides contact information. The page is annotated with HTML tags: <header> and </header> for the title and image; <nav> and </nav> for the navigation menu; <main> for the main content area; </main> for the end of the main content; and <footer> and </footer> for the contact information.

Page layout with sub-divided content (study page): example 2

Page content can be sub-divided into different areas using both <section> and <div> elements. Using both elements allows the sub-divided content to be independently styled, without the need to implement classes or ids:

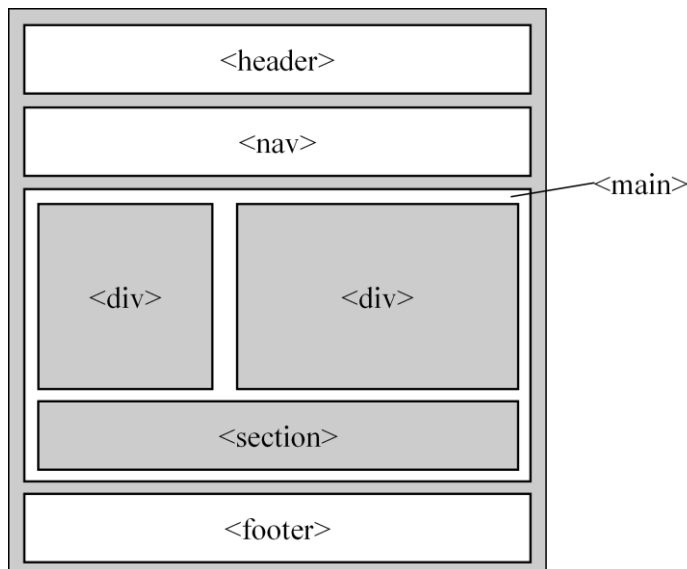


The following example website uses the study page to demonstrate the above layout:

The screenshot shows a website page for 'School Activities'. The page has a header with the title 'School Activities' and a navigation menu with links for 'Home', 'Sport', 'Music', 'Study', and 'Drama'. The main content area is divided into several sections: 'The Importance of Study', a news item about a 'study skills' seminar, a section for 'Learn more about study techniques' with three learner types (Visual, Tactile or Kinesthetic, Auditory), and a 'Contact Us' section. Annotations on the right side of the page point to specific HTML tags: `<main>` points to the main content area, `<section>` points to the 'The Importance of Study' section, `</section>` points to the end of the 'The Importance of Study' section, `<section>` points to the news item section, `</section>` points to the end of the news item section, `<section>` points to the 'Learn more about study techniques' section, `</section>` points to the end of the 'Learn more about study techniques' section, and `</main>` points to the end of the main content area.

Page layout with side-by-side content (drama page): example 3

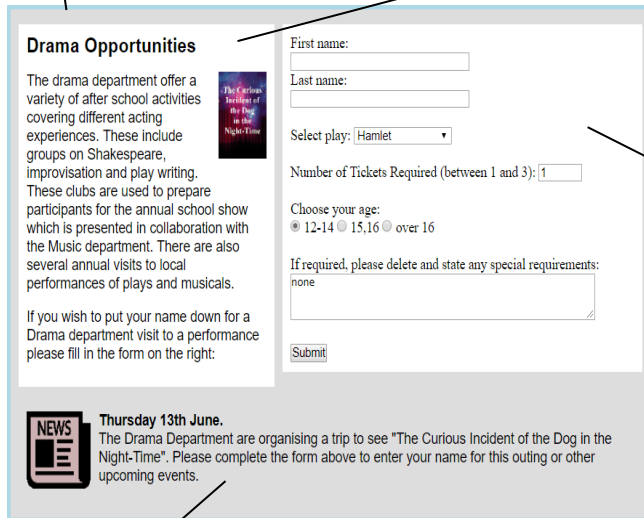
Page content can also be styled with CSS, to position page components side-by-side. In this example, the content has been sub-divided using three section elements, which are then controlled using float and clear CSS declarations:



The drama page shows the area containing the form, positioned to the right of the drama text area using CSS.

<main> — this is used to wrap around all the main content of the page. This is styled a light-grey colour:

<div> — the first div is styled white and uses `float:left` and `width:300px` to implement position and size.



<div> — the float left applied to the first div ensures that this area positions itself in line with the first. The margin between the two sections was created by pushing the left-hand edge of this section away from the left-hand edge of the page using `margin:330px`.

<section> — the style `clear:both` ensures that the final section sits on a new line and does not overlap the two sections above. The outer edge of the section is not apparent as it uses styles contained within the id 'newsArticle' to appear the same light-grey colour as the main element.

Appendix 16: HTML — forms (WDD)

The HTML `<form>` element defines a form that is used to collect user input:

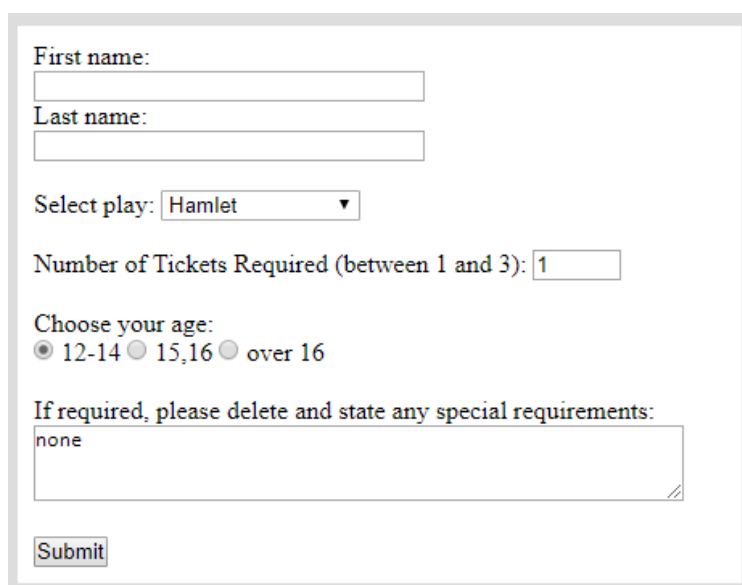
```
<form>
.
form elements
.
</form>
```

HTML 5 can be used to create and perform client-side validation on forms, without the need for JavaScript. Form elements implemented at Higher level are:

- ◆ input
 - text
 - number
 - textarea
 - radio
 - submit
- ◆ select

Where appropriate, form inputs will be validated using length, presence and range checks.

The following examples have been taken from the form used within the drama page of the *Higher Computing Science example website*. The example website can be downloaded as a zip file from the [Higher Computing Science](#) page on SQA's website. To view the full code in context, open and view the source code from the drama page.



The screenshot shows a web form with the following elements:

- Text input field for "First name:"
- Text input field for "Last name:"
- Dropdown menu for "Select play:" with "Hamlet" selected.
- Text input field for "Number of Tickets Required (between 1 and 3):" with the value "1" entered.
- Radio buttons for "Choose your age:" with options "12-14", "15,16", and "over 16". The "12-14" option is selected.
- Text area for "If required, please delete and state any special requirements:" with the text "none" entered.
- Submit button.

Input: example 1 — text

The example below implements two text input boxes, including length and presence checks:

```
<form>
First name:<br>
<input type="text" name="firstname" size="30" maxlength="15"
required><br>
Last name:<br>
<input type="text" name="lastname" size="30" maxlength="15"
required>
</form>
```

The above code includes the following:

<code>type="text"</code>	Identifies input type of the form element as text.
<code>name="firstname"</code>	The name attribute is required when a form's data is submitted to a server for processing. Although submitting a form to a server is not required until Advanced Higher, it is good practice to include the name attribute in all form elements.
<code>size="30"</code>	Width of the input box, in characters, when displayed in a browser.
<code>maxlength="15"</code>	Length check limiting input to 15 characters.
<code>required</code>	A presence check is applied to the input.

Input: example 2 — number

Number input may be limited to a minimum value, maximum value or both:

```
Number of Tickets Required (between 1 and 3):
<input type="number" name="tickets" min="1" max="3">
```

The above code includes the following:

<code>type="number"</code>	Identifies input type of the form element as numeric.
<code>min="1" max="3"</code>	A range check to ensure values entered are ≥ 1 and ≤ 3 .

Input: example 3 — textarea

A larger text box, for use with extended text input, can be implemented using the textarea form element:

If required, please delete and state any special requirements:
<textarea name="message" rows="3" cols="55"> </textarea>

The width and height of the textarea element is set using rows and columns.
If required, a length and presence check can be applied to the above input element.

Input: example 4 — radio buttons

Radio input can be implemented using multiple input elements of type radio:

```
Choose your age:<br>
<input type="radio" name="age" value="12 to 14"> 12-14
<br>
<input type="radio" name="age" value="15 or 16"> 15,16
<br>
<input type="radio" name="age" value="17 or over"> over 16
```

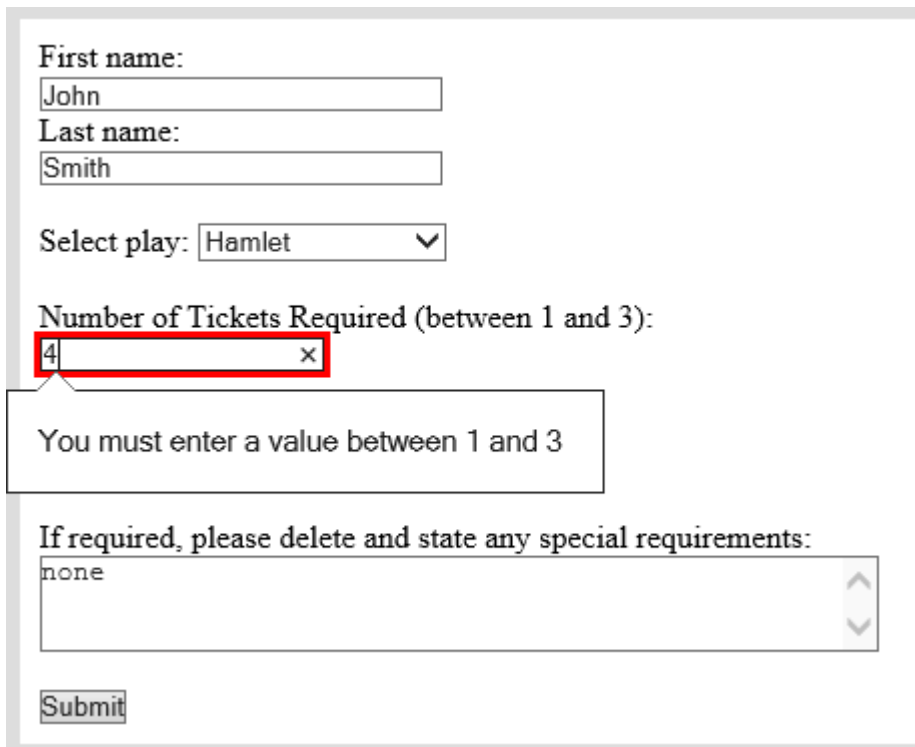
When submitted, the above form would return the name attribute “age” along with one of the listed values: 12 to 14, 15 or 16, 17 or over. Although Higher forms are not submitted to a server, it is good practice to include both the name and value attributes.

If the
 elements are omitted from the form, the radio buttons align horizontally.



Input: example 5 — submit

When a form is submitted, the browser shows any validation errors (check browser versions, as this is browser dependent).



The screenshot shows a web form with the following fields: "First name:" with the value "John", "Last name:" with the value "Smith", "Select play:" with a dropdown menu showing "Hamlet", and "Number of Tickets Required (between 1 and 3):" with a text input containing "4". A red border highlights the "4" in the text input, and a tooltip message above it says "You must enter a value between 1 and 3". Below the text input is a text area with the value "none" and a "Submit" button.

To allow candidates visual acknowledgement that an action is performed when the submit button is clicked, a JavaScript onclick event can be applied to the button as shown below:

```
<input type="submit" onclick="alert('Form Entered')" value="Submit">
```



Visual acknowledgement that the form is submitted can be helpful to candidates who are not yet experiencing a response generated by server-side processing.

Select: example 1 — drop-down menu

The select element is used to create a list of possible inputs in the form of a drop-down menu. Input choices are placed inside option elements:

Select play:

```
<select name="play">
  <option value="hamlet">Hamlet</option>
  <option value="godo">Waiting for Godo</option>
  <option value="brothers">Blood Brothers</option>
  <option value="curious">Curious Incident</option>
</select>
```

A screenshot of a web form. On the left, the text "Select play:" is followed by a dropdown menu. The dropdown menu is currently showing "Hamlet" and has a small downward-pointing arrow on its right side.

As previously stated, the name and value attributes are not required at Higher, but it is good practice to include both.

Select: example 2 — drop-down menu with size attribute

The size attribute can be used within the select element, to display a set number of options. If the number of options is larger than the size attribute, a scroll bar will automatically appear:

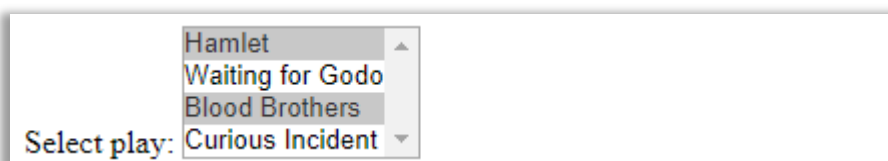
```
<select name="play" size="3">
```

A screenshot of a web form. On the left, the text "Select play:" is followed by a dropdown menu. The dropdown menu is open, showing three options: "Hamlet", "Waiting for Godo", and "Blood Brothers". The menu has a scroll bar on its right side.

Select: example 3 — drop-down menu with multiple attribute

To allow users to select more than one option, the multiple attribute is used with the select element:

```
<select name="play" size="4" multiple>
```

A screenshot of a web form. On the left, the text "Select play:" is followed by a dropdown menu. The dropdown menu is open, showing four options: "Hamlet", "Waiting for Godo", "Blood Brothers", and "Curious Incident". The menu has a scroll bar on its right side.

Pre-populating form input

To aid user input, form elements can be given values that are displayed when the web page loads. These can be left unchanged, deleted or edited by the user, when they are completing the form.

Example 1: text

The value attribute can be used to pre-populate text input elements:

```
<input type="text" name="firstname" value="forename"
size="30" maxlength="15" required>
```

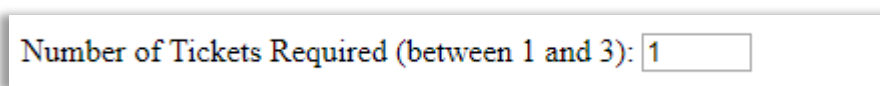


First name:
forename

Example 2: number

The value attribute is also used to pre-populate numeric input elements:

```
<input type="number" name="tickets" value="1" min="1"
max="3">
```

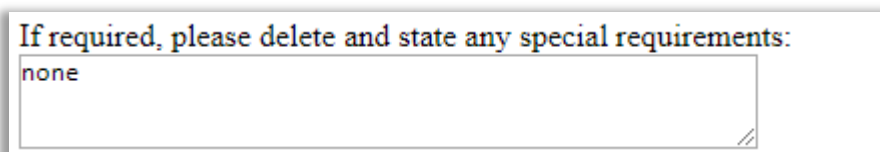


Number of Tickets Required (between 1 and 3): 1

Example 3: textarea

To pre-populate a textarea element, the text is included between the start and end elements:

```
<textarea name="message" rows="3" cols="55">none</textarea>
```



If required, please delete and state any special requirements:
none

Example 4: radio

Checked is used to initially check one of the radio buttons in a form:

```
<input type="radio" name="age" value="12 to 14" checked>
12-14 <br>
```



Choose your age:
 12-14 15,16 over 16

Appendix 17: JavaScript (WDD)

JavaScript events (onmouseover, onmouseout and onclick) are used to implement interactive web content.

Events are placed within HTML elements as shown below:

```
<img onmouseover=" " ">
```

The inverted commas contain the action to be executed when the event takes place. Possible actions include:

- ◆ hiding page elements
- ◆ revealing page elements
- ◆ changing the position of an element
- ◆ changing the size of an element
- ◆ changing the colour of an element
- ◆ changing the look of text

Actions can be executed by:

- ◆ referring to the element containing the JavaScript event
- ◆ referring to a different element, identified by an id
- ◆ calling a JavaScript function containing the actions

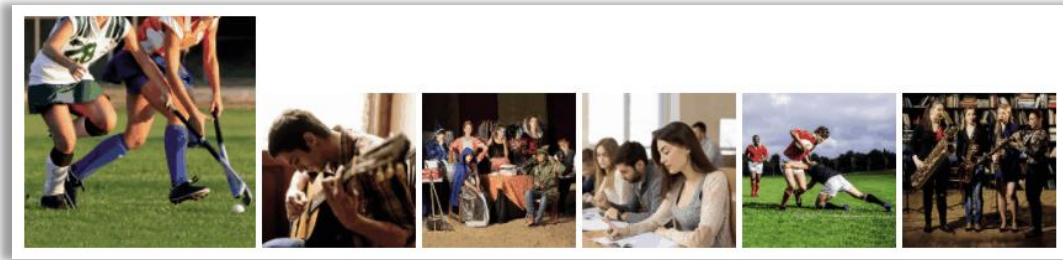
The following examples have been taken from the *Higher Computing Science example website*. The example website can be downloaded as a zip file from the [Higher Computing Science](#) page on SQA's website. To view the full code in context, open and view the source code from the pages noted in each example.

Interactively changing the size of a graphic (home page): example 1

Inline JavaScript can be used to increase and decrease the size of a graphic, as a mouse pointer passes over and out of a graphic.

```

```



This example contains two events, `onmouseover`, `onmouseout` used to execute four actions:

```
width='150px' height='150px' width='100px' height='100px'
```

Each action in a JavaScript statement is separated by a semi-colon.

Each action in the above events are prefixed by `this.style`.

`this` meaning this element. In example 1 this refers to the graphic (`img`) element containing the event.

`style` the style of this element will be altered using a CSS declaration

Interactively changing the size of a graphic (home page): example 2

JavaScript functions may be called to increase and decrease the size of a graphic, as a mouse pointer passes over and out of a graphic.

```

```

The JavaScript functions above are passed the parameter `this`, referring to the element containing the event `displayLarger(this)`

The JavaScript functions are placed within a `<script>` element in the `<head>` section of the HTML document:

```
<script>
function displayLarger(my_image)
    {my_image.style.width='150px';
    my_image.style.height='150px';}

function displaySmaller(my_image)
    {my_image.style.width='100px';
    my_image.style.height='100px';}
</script>
```

The parameter is used by each function to implement the same actions as example 1.

Implementing example 2 as a function, allows the same code to be called from multiple events:

```



```

Creating a rollover image (sports page): example 1

Rollover images can be created using two JavaScript events. The first event (`onmouseover`) displays an alternate graphic when the mouse passes over the image element. The second event (`onmouseout`) displays the original graphic when the mouse pointer leaves the image.

The following is the view in browser when the mouse pointer is not over any image:



The following is the view in browser when mouse passes over the right-hand image:



This can be implemented using the inline JavaScript shown below:

```

```

The action `src=' '` points the image element towards a given address.

Creating a rollover image (sports page): example 2

A rollover image can also be achieved using two functions. The first function is called using onmouseover. The second is called using onmouseout. Each function is passed the img element as a parameter using this. As before each function changes the src of the image:

HTML

```

```

JavaScript

```
<script>
function displaySport1A(my_image)
    {my_image.src='../images/netball.jpg';}
function displaySport1B(my_image)
    {my_image.src='../images/football.jpg';}
</script>
```

Highlighting text by dynamically changing its colour (home page):

Text may be highlighted by changing its style when the mouse pointer passes over the element containing the text.

The example below highlights a paragraph element dark red, by using an onmouseover event:

```
<p onmouseover="this.style.color='darkRed'">

Wednesday 12th June.<br> The Computing ... .. Room B9.
</p>
```

Highlighting text by dynamically changing its colour (study page):

To implement an action on a different page element, an id is required to identify the element the action will be performed on.

An element is identified using `document.getElementById(' ')`

The example below highlights the paragraph text when the mouse passes over the image contained in the paragraph:

```
<p id="highlight"><b>Thursday 13th June.</b><br> The guidance ...
... suits them. </p>
```

Revealing an element using onclick (study quiz page):

When a web page loads, elements may be initially hidden by applying the CSS declaration `display:none`. This can be implemented using an inline or internal style, or by implementing an external style (using a class, as shown below):

CSS

```
.hidden{display:none}
```

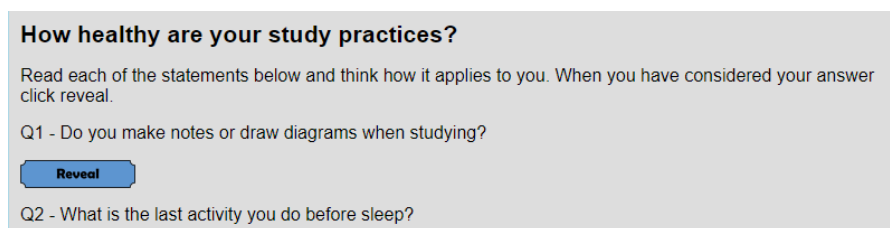
The graphic element in the example (reveal.png) contains an onclick event that uses the paragraph element's id to execute the action 'display=block' on the paragraph. The result of this action is that the paragraph becomes visible on the page.

HTML

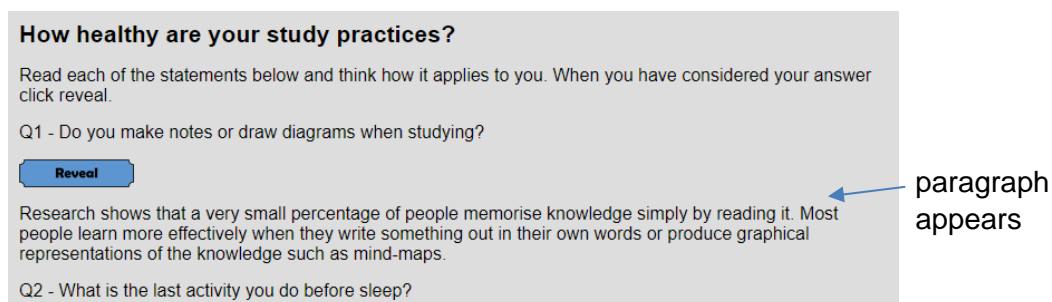
```
<p> Q1 - Do you make notes or draw diagrams when studying? </p>

<p id="reveall" class="hidden"> Research shows that a very small
percentage of people memorise knowledge simply by reading it. Most
people learn more effectively when they write something out in their
own words or produce graphical representations of the knowledge such
as mind-maps.</p>
```

The following is the view in browser when page loads:



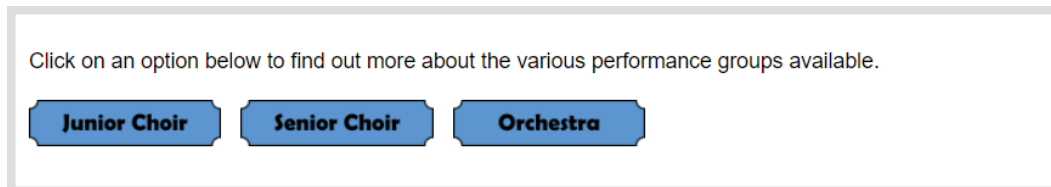
The following is the view in browser after 'Reveal' has been clicked:



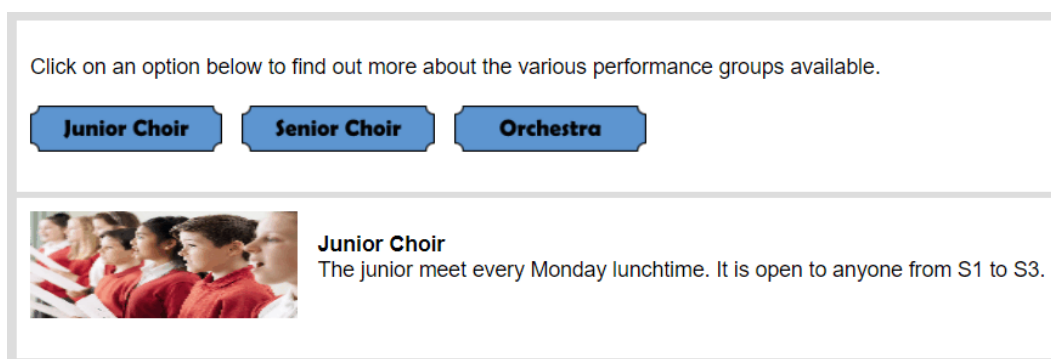
Revealing one of three hidden elements using onclick (music page):

Image elements, with associated onclick events, can be used to offer users a choice of what they view on a web page. Each onclick event is used to reveal a hidden <section> element containing a graphic and text relating to each option.

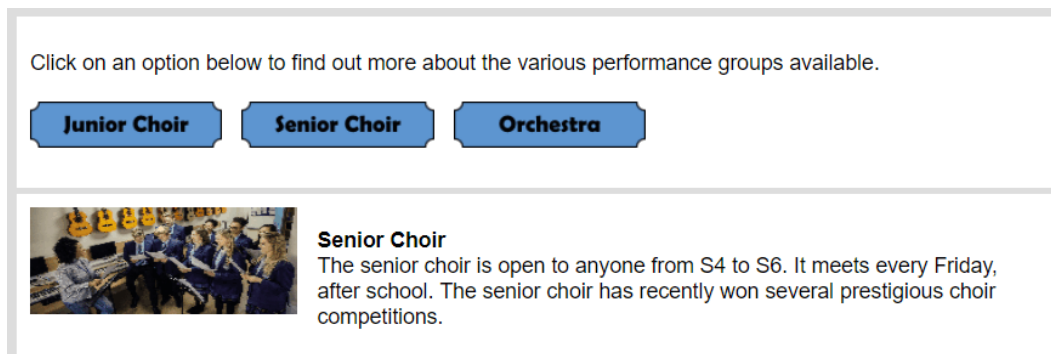
The following is the view in browser when page loads:



The following is the view in browser when the 'Junior Choir' graphic is then clicked.



The following is the view in browser when the 'Senior Choir' button is then clicked.



Each onclick event calls a function:

```
<p> Click on an option below to find out more about the
various performance groups available. <br><br>



</p>
```

The three section elements, containing the graphic and text to be revealed, are each identified by an `id` and initially hidden using `display:none`.

```
<section id="junior" style="display:none">

<p><b>Junior Choir</b><br>The junior meet every Monday
lunchtime. It is open to anyone from S1 to S3.</p>
</section>
<section id="senior" style="display:none">

<p><b>Senior Choir</b><br>The senior choir is open to anyone
from S4 to S6. It meets every Friday, after school. The
senior choir has recently won several prestigious choir
competitions.</p>
</section>
<section id="orchestra" style="display:none">

<p><b>Orchestra</b><br>The orchestra is not age limited,
instead being open to anyone who plays their instrument at
grade 3 level or higher.</p>
</section>
```

Each of the three JavaScript functions called by the `onclick` event reveals one of the three `<section>` elements and hides the other two using three `style.display` actions:

```
<script>
function displayJC() {
    document.getElementById("junior").style.display="block";
    document.getElementById("senior").style.display="none";
    document.getElementById("orchestra").style.display="none";
}

function displaySC() {
    document.getElementById("junior").style.display="none";
    document.getElementById("senior").style.display="block";
    document.getElementById("orchestra").style.display="none";
}

function displayO() {
    document.getElementById("junior").style.display="none";
    document.getElementById("senior").style.display="none";
    document.getElementById("orchestra").style.display="block";
}
</script>
```

Appendix 18: testing (WDD)

Usability testing

Usability testing involves systematic observation to determine how well people can use a product. In this case, the product will be the low-fidelity prototypes of a website. The goal with usability testing is to recreate real-world scenarios where the tester will actually be able to use your product. Then, by observing their behaviour, you will be able to understand what could be done better. In this case, the product will be the low-fidelity prototypes of a website. This helps to eliminate design problems at an early stage, before money has been spent implementing the design.

The testers may be given:

- ◆ a persona — this may relate to the age or experience that the tester should exhibit
- ◆ test cases — a set of actions executed to verify a particular feature or function of the website
- ◆ scenarios — they may be asked to use the website to place an order or book flights

They use the low-fidelity prototypes under a variety of conditions, while they are observed.

The observers make notes about any difficulties that the testers experienced and what alterations are required to the website design to make it easier to use the website.



Testing websites

There are a number of tests you should carry out on your website to ensure that it meets the functional requirements.

- ◆ Input validation:
 - check that every field in a form has the correct validation by trying to get every field on the form to accept incorrect data
- ◆ Links and navigation:
 - test the navigational bar links take you to the correct pages
 - test all external links work correctly
 - test that all pages can get back to the home page
 - test all internal links work correctly
 - test to check if there are any orphan pages (pages that are not linked to any others)
- ◆ Media content:
 - ensure that the text, graphics and video display correctly and in the position in which it was designed to appear

Compatibility testing

This is when you test your website to ensure that it works in the same way across a range of platforms.

Types of compatibility testing include:

Browser testing

It is important that your website will work on all the main browsers, for example Chrome, Firefox, Internet Explorer, Safari, and Opera. Your customers will not use your website if it does not function properly on their chosen browser.

Device type

You should check that your website is accessible on tablets, smartphones and desktop computers, as there are so many different types of hardware with different size screens available.

Common compatibility testing exposes the following types of problem:

- ◆ changes in font size
- ◆ changes in the user interface
- ◆ alignment issues
- ◆ changes in CSS style and colour
- ◆ scroll bar related issues
- ◆ content or label overlapping
- ◆ broken tables or frames

Copyright acknowledgements

All images Shutterstock:

Page 111,112,113,115,121 and 122: School activities banner — 553188940

Page 112,114 and 123: Drama opportunities — 329907647

Page 112,115,116,122 and 123: News icon — 735471220

Page 116 and 135: Junior choir — 432363805

Page 131: Hockey — 132311264

Page 131: Guitar — 145132324

Page 131: Drama — 752638612

Page 131: Studying — 549484090

Page 131: Rugby — 325167287

Page 131: Wind band — 723313330

Page 132: Footballer lifting trophy — 370092275

Page 132: Table tennis — 87611998

Page 132: Karate — 740133061

Page 132: Basketball — 198943043

Page 135: Senior choir — 588701573

Page 137: Man designing a website — 698323777

Administrative information

Published: May 2023 (version 3.0)

History of changes

Version	Description of change	Date
2.0	Table header on page 10 changed from 'Software' to 'Database'. Course support notes added as appendix.	June 2018
2.1	Appendix 5 'range and precision' section — references to 'accuracy' changed to 'precision'. Appendix 10 and appendix 11 — wildcard notations changed from Access specific (*) and (?) to generic (%) and (_), reflecting how they will be presented in the assessment. Appendix 12 — distorted graphics replaced.	August 2018
2.2	Page 101 (appendix 11 — SQL): we have amended the guidance on the UPDATE query to clarify the difference between National 5 and Higher.	October 2020
2.3	Page 37 — National 5 content changed from EU GDPR to UK GDPR, in line with change made to National 5 course specification.	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

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

© Scottish Qualifications Authority 2018, 2020, 2021, 2023