Curriculum & Assessment Map

Computer Science

TRANSITION STAGE

COMPUTER SCIENCE						
YEAR 7	**AUTUMN 1**	**AUTUMN 2**	**SPRING 1**	**SPRING 2**	**SUMMER 1**	**SUMMER 2**
CONTENT	**Induction** Understanding and using the school network Emailing Google Classrooms and the G Suite Effective internet searching E-safety Hardware and Software Networking Binary	**Control and Programming - GameMaker** Analysing games Designing a game using a model Designing sprites to add to game Creating a Game using Game Maker/GML language Testing game	**Control and Programming - GameMaker** Analysing games Designing a game using a model Designing sprites to add to game Creating a Game using Game Maker/GML language Testing game	**Dragons' Den -Desk Top Publishing and Finances** Creating a game cover with Desktop publishing Developing spreadsheet skills Understanding finances	**Dragons' Den -Advertising - Graphic development** Creating a logo using graphic software Adapting existing graphics to fit company theme	**Dragons' Den - Presentations** Creating a presentation to advertise a product Using video software to advertise product
SKILLS	- Understanding and effectively using the school network and G Suite tools effectively - Effective use of email and search technologies - Navigates the web and can carry out simple web searches to collect digital content. Explains the difference between a web browser and a search engine - Using technology responsibly, recognise acceptable/ unacceptable online behaviour - Understands the importance of	- Understanding of aim and audience - Use of gaming and programming concepts such as sprite, objects, events, actions, rooms - Use of a textual programming languages to solve a computational problem - Using logical reasoning to detect and correct errors	- Understanding of aim and audience - Use of gaming and programming concepts such as sprite, objects, events, actions, rooms - Use of a textual programming languages to solve a computational problem - Using logical reasoning to detect and correct errors	- Understanding of aim and audience - Development of DTP skills to create a gamecover - Understanding of typography skills - Development of spreadsheet skills and understanding to develop a financial model -Understanding of financial terms	- Understanding of aim and audience - Development of graphic bitmap software skills to create new graphics and adapt existing ones - Uses a variety of software to manipulate and present digital content: data and information. - Creates digital content to achieve a given goal through combining software packages and internet services to communicate with a wider audience	- Understanding of aim and audience - Development of presentation software skills to create an effective presentation - Development of video creations dn editing skills - Uses a variety of software to manipulate and present digital content: data and information. - Creates digital content to achieve a given goal through combining software packages and internet services to communicate with a wider audience

	communicating safely and respectfully online, and the need for keeping personal information private - understand the hardware and software components that make up computer systems, and how they communicate with one another and with other systems - Classifies a range of software including operating systems, utility and application software. Explains the difference between hardware and software, and their roles within a computer system. - Recognises and can use a range of input and output devices. Recognises that all software executed on digital devices is programmed. - Summarises the difference between the internet and internet service e.g. world wide web - understand how numbers can be represented in binary				- Shows an awareness for the quality of digital content collected. - Talks about their work and makes improvements to solutions based on feedback received. - Understanding of typography skills	- Shows an awareness for the quality of digital content collected. - Development of oracy skills
ASSESSMENT	Computer components, networks and binary assessment	Game development practical project	Game development practical project	Dragons Den Development Project Spreadsheet assessment In school synoptic exam	Dragons Den Development Project	Dragons Den Development Project

USEFUL RESOURCES/GUIDANCE:

- https://www.thinkuknow.co.uk/
- http://www.teach-ict.com/
- http://www.yoyogames.com/gamemaker/windows
- https://www.bbc.co.uk/bitesize/subjects/zvc9q6f
- https://idea.org.uk/

Curriculum & Assessment Map

FOUNDATION STAGE

COMPUTER SCIENCE						
YEAR 8	**AUTUMN 1**	**AUTUMN 2**	**SPRING 1**	**SPRING 2**	**SUMMER 1**	**SUMMER 2**
CONTENT	**Control and Programming** Understanding basic commands in programming environments such as LOGO.	**Control and Programming** Understanding basic commands in programming by using the Small BASIC programming environment to apply syntax	**Control and Programming - Python** - Understand the structural components of a program (variable declarations, command sequences, selection, iteration, data structures, subprograms) - Understand how to write code that responds appropriately to user input	**Data** - Data representation - Binary, HEX and ASCII Representation - Graphics and Sound representation - Data storage and compression	**Graphics creation** Understanding and creating different types of graphics solution to small real-world examples.	**HTML and CSS** Website development using HTML, CSS and Javascript
SKILLS	Understanding and applying syntax and errors	Applying syntax and sequencing, programming constructs and variables and understanding algorithms and pseudocode Understanding and applying syntax and errors, using concatenation and datatypes, using different constructs in programming, understanding sub-procedures and functions, applying all skills into a project	Understanding and applying syntax and errors. Be able to use different constructs in programming, understanding the use of variable, sub-procedures and functions, applying all skills into a project	- Understand that computers use binary to represent data (numbers, text, sound, graphics) - Be able to convert binary, hexadecimal and between the terms 'bit, nibble, byte, kilobyte (KB), megabyte (MB), gigabyte (GB), terabyte (TB)' - Understand how computers encode characters [ASCII] - Understand how bitmap images and sound are represented in binary - Understand that file storage is measured in bytes and be able to calculate file sizes - Understand how a lossless, run-length encoding algorithm works	Understanding vector and bitmap graphics, using Photoshop software and different tools to create and edit graphics	- Understand what HTML and CSS are used for - Be able to explain and implement HTML tags - Be able to define hyperlinks, explain their use and create a link using tags • Be able to describe what inline CSS is and use style attributes - Use CSS stylesheet in a high-level language - Be able to understand and implement javascript into a webpage

Curriculum & Assessment Map

ASSESSMENT	LOGO programming assessment	Small BASIC programming assessment	Python programming assessment In school synoptic exam	Data representation assessment	Graphic development practical assessment	HTML and websites assessment
USEFUL RESOURCES/GUIDANCE: • http://www.codecademy.com/ • https://www.python.org/ • http://www.bbcbasic.co.uk/bbcbasic.html • http://www.teach-ict.com/ • https://www.bbc.co.uk/bitesize/subjects/zvc9q6f • https://www.w3schools.com/ • https://idea.org.uk/						

COMPUTER SCIENCE						
YEAR 9	**AUTUMN 1**	**AUTUMN 2**	**SPRING 1**	**SPRING 2**	**SUMMER 1**	**SUMMER 2**
CONTENT	**E-safety and Legal issues of IT** - Understand how to use technology responsibly and recognising acceptable/unacceptable online behaviour - Understanding the legal and ethical considerations when working with technology **Decomposition and abstraction** - Understand how to and be able to decompose a problem - Understand how to and be able to analyse, design and program a solution to small real-world examples	**Algorithms** - Understanding different types of algorithms - Creating algorithms using programming constructs - Output from algorithms - Identifying errors in algorithms - Coding using algorithms - Algorithms to sort and search	**Programming** - Understand the structural components of a program (variable declarations, command sequences, selection, iteration, data structures, subprograms) - Understand how to write code that responds appropriately to user input - Understand the purpose and use of arithmetic, relational and logic operators	**Programming** - Understand how to and be able write programs in a high-level programming language - Understand the, need use and benefits of using subprograms	**Computers and Software** - Components of a computer - Fetch-decode-execute cycle - Primary storage - Secondary storage - Operating Systems - Utility software	**Networks and Network Security** - Computer networks - Network topologies - Network protocols - Network layers - Internet technologies - Network security
SKILLS	- Be aware of the risks and e-safety issues and impacts of working with technology	- Be able to use and interpret algorithms (flowcharts, pseudocode,	- Be able to use sequencing, selection and iteration constructs in their programs	- Produce programs that are easy to read and be able to use techniques (comments,	- Identify the components of a computer with respect	- Understand the different types of networks

	- Be aware of ethical and legal issues arising from the use of computers - Be able to analyse a problem, investigate requirements (inputs, outputs, processing, initialisation) and design solutions - Be able to decompose a problem into smaller sub-problems - Understand how abstraction can be used effectively to model aspects of the real world - Be able to program abstractions of real-world examples	written descriptions, program code) - Create algorithms to solve a particular problem, making use of programming constructs (sequence, selection, iteration) and using appropriate conventions (flowchart, pseudo-code, written description, draft program code) - Determine the correct output of an algorithm for a given set of data - Identify and correct errors in algorithms - Code an algorithm in a high-level language - Use and implement sorting and searching algorithms (bubble sort, merge sort, linear search, binary search)	- Be able to use, data types (integer, real, Boolean, char) and data structures (records, one-dimensional arrays, two-dimensional arrays) - Be able to manipulate strings - Use global/local variables and constants in programming - Be able to write code that reads/writes from/to a text file - Use arithmetic operators (add, subtract, divide, multiply, modulus, integer division) - Use relational operators (equal to, less than, greater than, not equal to, less than or equal to, greater than or equal to) - Use logic operators (AND, OR, NOT)	descriptive names (variables, constants, subprograms), indentation) to improve readability - Be able to differentiate between types of error in programs (logic, syntax, runtime) - Be able to design and use test plans and test data (normal, boundary, erroneous) - Be able to interpret error messages and identify, locate and fix errors in a program - Be able to determine what value a variable will hold at a given point in a program - Be able to determine the strengths and weaknesses of a program and improve it - Be able to write code that uses user-written and pre-existing subprograms, procedures, and functions - Be able to create subprograms that use parameters	to: input devices, output, storage devices - Be able to explain the role of the fetch-decode-execute cycle. - Identify the hardware components used in the von Neumann architecture and explain their role in the fetch-decode-execute cycle - Be able to explain the purpose of RAM, Cache and ROM - Be able to explain the purpose and key characteristics of secondary storage such as Magnetic storage, Optical storage and Flash Memory - Identify a range of operating systems, including Open Source and Proprietary - Be able to explain the operating systems are needed and the basic functions of an operating systems - Be able to explain the purpose of different system utilities	- Identify the hardware components required for networking - Identify the different arrangements of various devices in a network - Be able to explain the advantages and disadvantages of each type of topology - Understand how devices are addressed in networks - Be able to explain the various protocols used in networking - Understand the concept of layering and how it is applied to networks - Understand what the internet is and the different technologies and applications it has - Be able to explain the Client-server model - Identify misleading websites - Can determine threats to computers such as malware, phishing and brute-force attack
ASSESSMENT	E-Safety and ICT Laws assessment	Algorithm development assessment	Python programming assessment In school synoptic exam	Python programming assessment	Computer components (Hardware) assessment	Networking assessment
USEFUL RESOURCES/GUIDANCE: • http://www.codecademy.com/ • https://www.python.org/ • http://www.teach-ict.com/ • https://www.bbc.co.uk/bitesize/subjects/zvc9q6f •						

Curriculum & Assessment Map

EXAMINATION STAGE

COMPUTER SCIENCE						
YEAR 10	**AUTUMN 1**	**AUTUMN 2**	**SPRING 1**	**SPRING 2**	**SUMMER 1**	**SUMMER 2**
CONTENT	**Decomposition and abstraction** -Modelling aspects of the real world -Subprograms **Algorithms** -Flowcharts, pseudocode and program code -Constructs and input, processing and output -Variables and 1D and 2D structures -Arithmetic and logical operators -Errors in algorithms **Developing code** -Reading and writing programs in a high-level programming language -Converting algorithms -Programming errors **Programming Constructs** -Sequencing, selection, repetition **Data types and structures** -Primitive data types and 1D and 2D structures - Variables and constants -String Manipulation **Input/output** - Programs that respond to user input **Operators** -Arithmetic, relational and logical operators **Subprograms**	**Algorithms** -Flowcharts, pseudocode and program code -Constructs and input, processing and output to solve problems -Variables and 1D and 2D dimensional data structures -Arithmetic and logical operators **Truth tables** -Using truth tables to solve problems **Developing code** -Decomposition and abstraction used to analyse, understand and solve problems -Reading and writing programs in a high-level programming language -Converting algorithms into programs **Programming Constructs** -Structural components of programs -Sequencing, selection, repetition **Data types and structures** -Primitive data types and 1D and 2D-dimensional structured data types **Operators** Arithmetic, relational and logical operators **Binary**	**Decomposition and abstraction** - Using decomposition and abstraction to model aspects of the real world -Subprograms **Algorithms** Flowcharts, pseudocode and program code -Constructs and input, processing and output -Variables and 1D and 2D data structures **Developing code** -Decomposition and abstraction used to analyse, understand and solve problems -Reading and writing programs in a high-level programming language **Data types and structures** -Primitive data types and 1D and 2D structured data types **Subprograms** -Pre-existing and user-devised procedures and functions -Functions and parameters **Hardware** -CPU and fetch-decode-execute cycle -RAM and ROM -Secondary storage	**Decomposition and abstraction** - Using decomposition and abstraction to model aspects of the real world -Subprograms **Algorithms** -Flowcharts, pseudocode and program code -Constructs and input, processing and output -Variables and 1D and 2D data structures -Algorithm outputs using trace tables -Sorting and Searching algorithms **Programming Constructs** -Sequencing, selection, repetition -Subprograms **Data types and structures** -Primitive data types and 1D and 2D structured data types -String manipulation **Input/output in programming** -Programs that accept and respond to user input -Validation in programming **Subprograms** -Pre-existing and user-devised procedures and functions	**Algorithms** -Flowcharts, pseudocode and program code -Constructs and input, processing and output -Variables and 1D and 2D data structures -Sorting and Searching algorithms **Programming Constructs** -Sequencing, selection, repetition **Data types and structures** -String manipulation **Input/output in programming** -Programs that accept and respond to user input -Reading and writing to csv and text files -Validation in programming -Authentication in programming **Software** -Utility software and their functionality -Robust software and identifying vulnerabilities **Network security** -Network security -Identifying network vulnerabilities and protecting networks -Threats to digital systems -Malware, hackers and cyberattacks	**Develop code** -Programming errors **Programming Constructs** -Sequencing, selection, repetition -Subprograms **Data types and structures** --Primitive data types and 1D and 2D structured data types -Variables and constants **Input/output in programming** -Programs that accept and respond to user input **Operators** -Arithmetic, relational and logical operators **Subprograms** -Pre-existing and user-devised procedures and functions -Functions and parameters -Global and local variables **Networks** -Different types of networks (LAN, WAN) -Structure of the internet -Wired and wireless connectivity -Network speeds -Network protocols -Network topologies

	Pre-existing and user-devised procedures and functions **Binary** -Binary representation of numbers -Binary arithmetic **Data representation, storage and compression** -Limitations of binary representation -Data storage, file calculation and data capacity requirements	-Representation of text, sound and graphics in binary -Binary arithmetic -Hexadecimal **Data representation, storage and compression** -ASCII -Image and sound data representation -Limitations of binary representation -Data storage, file calculation and data capacity requirements -Data compression		Functions and parameters **Software** -Operating systems and their functions (file management, process management, peripheral management, user management) -Utility software and their functionality	-Protecting of digital systems and data	
SKILLS	-Use decomposition and abstraction to model aspects of the real world -Follow and write algorithms (flowcharts, pseudocode*, program code) that use sequence, selection, repetition and iteration and input, processing and output -Follow and write algorithms that use variables and constants and 1D and 2D structures -Follow and write algorithms that use arithmetic, relational and logical operators -Identify the types of errors that can occur in programs and be able to identify and correct logic errors in algorithms	-Follow and write algorithms (flowcharts, pseudocode*, program code) that use sequence, selection, repetition and iteration and input, processing and output -Follow and write algorithms that use variables and constants and 1D and 2D dimensional data structures -Follow and write algorithms that use arithmetic, relational and logical operators -Apply logical operators in truth tables with up to three inputs -Use decomposition and abstraction to analyse, understand and solve problems -Read, write, analyse and refine programs	-Use decomposition and abstraction to model aspects of the real world and analyse, understand and solve problems -Understand the benefits of using subprograms -Follow and write algorithms (flowcharts, pseudocode*, program code) that use sequence, selection, repetition and iteration and input, processing and output to solve problems -Follow and write algorithms that use variables and constants and 1D and 2D data structures -Read, write, analyse and refine programs written in a high-level programming language -Write programs that make appropriate use of primitive data types	-Use decomposition and abstraction to model aspects of the real world and analyse, understand and solve problems -Understand the benefits of using subprograms -Follow and write algorithms (flowcharts, pseudocode*, program code) that use sequence, selection, repetition and iteration and input, processing and output to solve problems -Follow and write algorithms that use variables and constants and 1D and 2D data structures -Determine the correct output of an algorithm using a trace table -Be able to explain and use different search and sort algorithms (bubble	-Follow and write algorithms (flowcharts, pseudocode*, program code) that use sequence, selection, repetition and iteration and input, processing and output to solve problems -Follow and write algorithms that use variables and constants and 1D and 2D dimensional data structures --Be able to explain and use different search and sort algorithms (bubble sort, merge sort, linear search, binary search) -Write programs that make appropriate use of sequencing, selection, repetition (count-controlled, condition-controlled), iteration (over every item in a data structure)	-Identify, locate and correct program errors (logic, syntax, runtime) -Write programs that make appropriate use of sequencing, selection, repetition (count-controlled, condition-controlled), iteration (over every item in a data structure) and single entry/exit points from code blocks and subprograms -Write programs that make appropriate use of primitive data types (integer, real, Boolean, char) 1D and 2D structured data types (string, array, record) -Write programs that make appropriate use of variables and constant -Write programs that accept and respond appropriately to user input

	-Use decomposition and abstraction to analyse, understand and solve problems -Read, write, analyse and refine programs written in a high-level programming language -Convert algorithms into programs -Use techniques (layout, indentation, comments, meaningful identifiers, white space) to make programs easier to read, understand and maintain -Identify, locate and correct program errors -Identify the function and structural components of programs -Write programs that make appropriate use of sequencing, selection, repetition iteration and single entry/exit points from code blocks and subprograms -Write programs that make appropriate use of primitive data types and 1D and 2D data types -Write programs that make appropriate use of variables and constants -Write programs that manipulate strings -Write programs that accept and respond	written in a high-level programming language -Convert algorithms into programs -Identify the structural components of programs -Write programs that make appropriate use of sequencing, selection, repetition iteration and single entry/exit points from code blocks and subprograms -Write programs that make appropriate use of primitive data types and 1D and 2D structured data types -Write programs that use relational and logical operators - Understand that computers use binary to represent data (numbers, text, sound, graphics) and program instructions and be able to determine the maximum number of states that can be represented by a binary pattern of a given length -Be able to add together two positive binary patterns and apply logical and arithmetic binary shifts -Understand why hexadecimal notation is used and be able to convert between hexadecimal and binary	and 1D and 2D structured data types -Write programs that use pre-existing and user-devised subprograms (procedures, functions) -Understand the von Neumann stored program concept and the role of main memory (RAM), CPU clock, address bus, data bus, control bus in the fetch-decode-execute cycle -Understand the role of secondary storage and the ways in which data is stored on devices (magnetic, optical, solid state) -Be able to construct expressions to calculate file sizes and data capacity requirements	sort, merge sort, linear search, binary search) -Write programs that make appropriate use of sequencing, selection, repetition, iteration and single entry/exit points from code blocks and subprograms -Write programs that make appropriate use of primitive data types and 1D and 2D structured data types -Write programs that manipulate strings (length, position, substrings, case conversion) -Write programs that accept and respond appropriately to user input -Write programs that implement validation -Write programs that use pre-existing and user-devised subprograms (procedures, functions) -Understand the purpose and functionality of an operating system (file management, process management, peripheral management, user management) -Understand the purpose and functionality of utility software (file repair, backup, data compression, disk	and single entry/exit points from code blocks and subprograms -Write programs that manipulate strings (length, position, substrings, case conversion) -Write programs that read from and write to comma separated value text files -Write programs that implement validation (length check, presence check, range check, pattern check) -Write programs that implement authentication (ID and password, lookup) -Understand the purpose and functionality of utility software (file repair, backup, data compression, disk defragmentation, anti-malware) -Understand the importance of developing robust software and methods of identifying vulnerabilities (audit trails, code reviews) -**U**nderstand the importance of network security, ways of identifying network vulnerabilities (penetration testing, ethical hacking) and methods of protecting networks (access control, physical security, firewalls)	-Write programs that use arithmetic, relational and logical operators -Write programs that use pre-existing (built-in, library) and user-devised subprograms (procedures, functions) -W rite programs that make appropriate use of global and local variables -Understand why computers are connected in a network -Understand different types of networks (LAN, WAN) -Understand how the internet is structured (IP addressing, routers) -Understand how the characteristics of wired and wireless connectivity impact on performance (speed, range, latency, bandwidth) -Understand that network speeds are measured in bits per second (kilobit, megabit, gigabit) and be able to construct expressions involving file size, transmission rate and time -Understand the role of and need for network protocols (Ethernet, Wi-Fi, TCP/IP, HTTP, HTTPS, FTP) and email protocols (POP3, SMTP, IMAP)

	appropriately to user input -Write programs that use arithmetic operators -Be able to write programs that use pre-existing and user-devised subprograms (procedures, functions) -Understand that computers use binary to represent data and program instructions and be able to determine the maximum number of states that can be represented by a binary pattern of a given length -Understand how computers represent and manipulate unsigned integers and two's complement signed integers -Be able to convert between denary and 8-bit binary numbers -Be able to add together two positive binary patterns and apply logical and arithmetic binary shifts -Understand the concept of overflow in relation to the number of bits available to store a value -Understand the limitations of binary representation of data when constrained by	-Understand how computers encode characters using 7-bit ASCII -Understand how bitmap images are represented in binary -Understand how analogue sound is represented in binary -Understand the limitations of binary representation of data when constrained by the number of available bits **-U**nderstand that data storage is measured in binary multiples (bit, nibble, byte, kibibyte, mebibyte, gibibyte, tebibyte) and be able to construct expressions to calculate file sizes and data capacity requirements -Understand the need for data compression and methods of compressing data)		defragmentation, anti-malware)	**-U**nderstand the threat to digital systems posed by malware (viruses, worms, Trojans, ransomware, key loggers) and how hackers exploit technical vulnerabilities (unpatched software, out-of-date anti-malware) and use social engineering to carry out cyberattacks -Understand methods of protecting digital systems and data (anti-malware, encryption, acceptable use policies, backup and recovery procedures)	-Understand characteristics of network topologies (bus, star, mesh) -Be able to construct expressions to calculate file sizes and data capacity requirements

	the number of available bits -Understand that data storage is measured in binary multiples and be able to construct expressions to calculate file sizes and data capacity requirements					
ASSESSMENT	In class practice paper questions on topics of: -Decomposition and Algorithms -Binary Representation	In class practice paper questions on topics of: -Algorithms and Programming -Data Representation	In class practice paper questions on topics of: -Algorithms and Programming -Computer Hardware In school synoptic exam	In class practice paper questions on topics of: -Algorithms and Programming -Computer Software	In class practice paper questions on topics of: -Algorithms and Programming -Computer Software	In class practice paper questions on topics of: -Algorithms and Programming -Networks and Network security
USEFUL RESOURCES/GUIDANCE: • https://qualifications.pearson.com/content/dam/pdf/GCSE/Computer%20Science/2020/specification-and-sample-assessments/GCSE_L1_L2_Computer_Science_2020_Specification.pdf • http://www.codecademy.com/ • https://www.python.org/ • http://www.teach-ict.com/ • https://www.youtube.com/channel/UC0HzEBLIJxlrwBAHJ5S9JQg/playlists?view=50&sort=dd&shelf_id=21 • https://computerscienceuk.com/gcse-9-1/gcse-videos/ •Edexcel GCSE (9-1) Computer Science Student Book by Ann Weidmann, David Waller, Cynthia Selby (ISBN: 9781292359991)						

Curriculum & Assessment Map

COMPUTER SCIENCE						
YEAR 11	**AUTUMN 1**	**AUTUMN 2**	**SPRING 1**	**SPRING 2**	**SUMMER 1**	**SUMMER 2**
CONTENT	**Computational Thinking** -Decomposition and Abstraction - Understanding different types of algorithms - Creating algorithms using programming constructs - Output from algorithms - Identifying errors in algorithms - Coding using algorithms - Algorithms to sort and search - Testing algorithms **Binary and Data Representation** - Binary - Data representation - Data storage and compression - Encryption	**Problem solving with Programming** - Understand how to and be able write programs in a high-level programming language - Understand the structural components of a program (variable and type declarations, command sequences, selection, iteration, data structures, subprograms) - Understand how to convert algorithms into program code - Understand how to test program code - Understand the different types of errors **Computers and Software** - Computer Hardware - Logic - Systems Software - Applications Software - Programming languages	**Problem solving with Programming** - Understand the different types of data types and structures used in programming - Understand how to write programs that manipulate strings - Understand how to write programs that respond to user input - Understand different types of validations and authentication and how to write programs that can implement them **Networks and Network Security** - Networking - Types of networks - Connectivity and speed - Topologies - Transmission and protocols - Network Security	**Problem solving with Programming** - Understand the different types of operator (arithmetic, relational, logical) - Understand the, need use and benefits of using subprograms **Issues and impact** - Environmental - Ethical - Cybersecurity	**Revision and Practical exam practice**	**Study Leave**
SKILLS	**Computational Thinking**	**Problem solving with Programming**	**Problem solving with Programming**	**Problem solving with Programming**		

	- Understand the benefit of using decomposition and abstraction - Understand the benefits of using subprograms - Be able to follow and write algorithms that use sequence, selection, repetition and iteration - Understand the need for and be able to follow and write algorithms that use variables, constants, one- and two-dimensional data structures - Understand the need for and be able to follow and write algorithms that use arithmetic, relational and logical operators - Be able to determine the correct output of an algorithm for a given set of data - Understand types of errors that can occur in programs and be able to identify and correct logic errors - Understand how standard sorting and searching algorithms work - Be able to apply logical operators in truth tables **Binary and Data Representation** - Understand that computers use binary to represent data	- Be able to read, write, analyse and refine programs written in a high-level programming language - Be able to convert algorithms into programs -Be able to make programs easier to read, understand and maintain - Be able to identify, locate and correct program errors (logic, syntax, runtime) - Understand and be able to identify the structural components of programs (constants, variables, initialisation and assignment statements, command sequences, selection, repetition, iteration, data structures, subprograms, parameters, input/output) - Be able to write programs that make appropriate use of sequencing, selection, repetition and subprograms **Computers and Software** - Understand the von Neumann stored program concept and the role of main memory (RAM), CPU clock, address bus, data bus, control bus in the fetch-decode-execute cycle	- Be able to write programs that make appropriate use of primitive data types and one and two-dimensional structured data types - Be able to write programs that make appropriate use of variables and constants - Be able to write programs that manipulate strings - Be able to write programs that accept and respond appropriately to user input - Be able to write programs that read from and write to comma separated value text files - Understand the need for and be able to write programs that implement validation - Understand the need for and be able to write programs that implement authentication **Networks and Network Security** - Understand why computers are connected in a network - Understand different types of networks (LAN, WAN) - Understand how the internet is structured - Understand how the characteristics of wired and wireless	- Be able to write programs that use arithmetic, relational and logical operators - Be able to write programs that use pre-existing (built-in, library) and user-devised subprograms - Be able to write functions that may or may not take parameters but must return values, and procedures that may or may not take parameters but do not return values - Understand the difference between and be able to write programs that make appropriate use of global and local variables **Issues and impact** - Understand environmental issues associated with the use of digital devices - Understand ethical and legal issues associated with the collection and use of personal data - Understand ethical and legal issues associated with the use of artificial intelligence, machine learning and robotics - Understand methods of intellectual property protection for		

	(numbers, text, sound, graphics) and program instructions - Understand how computers represent and manipulate unsigned integers and two's complement signed integers - Be able to convert between denary and 8-bit binary numbers - Be able to add together two positive binary patterns and apply logical and arithmetic binary shifts - Understand the concept of overflow - Understand why hexadecimal notation is used and be able to convert between hexadecimal and binary - Understand how bitmap images are represented in binary - Understand how analogue sound is represented in binary - Understand the limitations of binary representation of data -Understand that data storage is measured in binary multiples and be able to construct expressions to calculate file sizes and data capacity requirements - Understand the need and methods of data compression	- Understand the role of secondary storage and the ways in which data is stored on devices (magnetic, optical, solid state) - Understand the concept of an embedded system - Understand the purpose and functionality of an operating system and utility software - Understand the importance of developing robust software and methods of identifying vulnerabilities - Understand the characteristics and purposes of low-level and high-level programming languages - Understand how an interpreter differs from a compiler	connectivity impact on performance - Understand that network speeds are measured in bits per second and be able to construct expressions involving file size, transmission rate and time - Understand the role of and need for network protocols - understand how the 4-layer TCP/IP model handles data transmission over a network -Understand characteristics of network topologies - Understand the importance of network security, ways of identifying network vulnerabilities and methods of protecting networks	computer systems and software - Understand the threat to digital systems posed by malware and how hackers exploit technical vulnerabilities and use social engineering to carry out cyberattacks - Understand methods of protecting digital systems and data		
ASSESSMENT	In class practice paper questions on topics of:	In class practice paper questions on topics of:	In class practice paper questions on topics of:	In class practice paper questions on topics of:	Walking talking practice exams both for both paper 1 and paper	

	-Computational Thinking -Data representation	-Problem solving with Programming (this may include some online practical questions) -Computers and Software In school synoptic exam	-Problem solving with Programming (this may include some online practical questions) -Networks and Network Security	-Problem solving with Programming (this may include some online practical questions) -Issues and Impact	2 (online practical exam)	

USEFUL RESOURCES/GUIDANCE:

https://qualifications.pearson.com/content/dam/pdf/GCSE/Computer%20Science/2020/specification-and-sample-assessments/GCSE_L1_L2_Computer_Science_2020_Specification.pdf

- http://www.codecademy.com/
- https://www.python.org/
- http://www.teach-ict.com/
- https://www.youtube.com/channel/UC0HzEBLIJxlrwBAHJ5S9JQg/playlists?view=50&sort=dd&shelf_id=21
- https://computerscienceuk.com/gcse-9-1/gcse-videos/
- Edexcel GCSE (9-1) Computer Science Student Book by Ann Weidmann, David Waller, Cynthia Selby (ISBN: 9781292359991)

Curriculum & Assessment Map

ADVANCED STAGE

AS LEVEL COMPUTER SCIENCE						
YEAR 12	AUTUMN 1	AUTUMN 2	SPRING 1	SPRING 2	SUMMER 1	SUMMER 2
CONTENT	**Component 1.1:** The characteristics of contemporary processors Input, output and storage devices **Component 1.2.3:** Introduction to programming **Component 2.2.1:** Programming techniques	**Component 1.2.1:** Operating Systems **Component 1.3.1:** Databases **Component 1.2.3:** Assembly language **Component 1.4.2:** Data structures **Component 2.3:** Algorithms	**Component 1.4.1:** Data Types **Component 1.5.1:** Computing related legislation **Component 1.2.3:** Assembly language **Component 2.3:** Algorithms **Component 2.1:** Elements of computational thinking	**Component 1.5.2:** Ethical, moral and cultural issues **Component 1.3.2:** Networks **Component 1.2.2:** Applications generation **Component 1.3.3:** Web Technologies **Component 2.1:** Elements of computational thinking **Component 1.3.3:** Boolean Algebra	**Component 1.2.2:** Applications generation **Component 1.3.3:** Boolean Algebra **Component 2.2.2:** Software Development	**Component 3.1:** Analysis of the problem **Component 1.2.4:** Types of Programming Language. **Component 3:** Introduction to Monkey-x
SKILLS	Develop deeper knowledge and understanding of: -Structure and function of the processor -Types of processor -Input, output and storage Practical work: -Write code in Python and Visual Basic to solve problems. Develop knowledge on programming structure such as: -Programming constructs -Global and local variables Modularity -Functions and procedures -Parameter passing by value and reference -Use of an IDE to develop/debug a program	Develop deeper knowledge and understanding of: -The need for, function and purpose of operating systems. -Memory Management -Interrupts Service Routines (ISR) and its role within the fetch decode execute cycle. -Scheduling - BIOS Develop knowledge and understanding of: -Relational database -flat file and how to capture, manage and exchange data. Practical Work: -Write code in Assembly language -Write code in python using data structures to solve problem. -Write algorithms to solve simple problems.	Develop knowledge and understanding and carry out activities on: - Represent positive integers in binary. -Use of sign and magnitude and two's complement to represent negative numbers in binary. -Addition and subtraction of binary integers. -Represent positive integers in hexadecimal -Convert Hexadecimal and binary and denary. -Positive and negative real numbers using normalised floating point representation. -How character sets (ASCII and UNICODE) are used to represent text. -data protection, data misuse, copyright and investigatory powers.	Develop knowledge and understanding of: -Legal, moral, ethical and cultural issues. - Characteristics of networks and the importance of protocols and standards. -Internet structure: The TCP/IP Stack. DNS -Protocol layering. -LANs and WANs. -Packet and circuit switching. -Client-server and Peer to peer. Develop knowledge and understanding of: -The nature of applications, justifying suitable applications for a specific purpose. -Utilities. Practical work to develop website using: - html, CSS and Java Script.	Develop knowledge and understanding of: -Waterfall lifecycle, agile methodologies, extreme programming, the spiral model and rapid application development. -The relative merits and drawbacks of different methodologies and when they might be used. - Writing and following algorithms. - Different test strategies, including black and white box testing and alpha and beta testing. -Test programs that solve problems using suitable test data and end user feedback, justify a test strategy for a given situation. Develop knowledge and understanding of:	-Investigate the features available on sample games and record the features it offers. -Research and find a suitable problem(game) to make a start on the project. -Define the stakeholder for the problem. Develop knowledge and understanding of: -Need for and characteristics of a variety of programming paradigms. -Procedural languages. -Object-oriented languages Practical work on Monkey-x programming language to develop games.

			-searching and sorting algorithms. - standard algorithms for add, delete items from stack and queue -suitability of algorithms for solving a problem. Develop knowledge and understanding of: Thinking abstractly and Thinking ahead	Develop knowledge and understanding of Lossy v lossless compression.	-Open source vs Closed source. -Translators: interpreters, compilers and assemblers. -Boolean expressions, Karnaugh maps.	
ASSESSMENT	Weekly tests based on topics taught	November internal exam Weekly tests based on topics taught	February internal exam Weekly tests based on topics taught	Weekly tests based on topics taught	End of year internal exam	Computing Project (Component 3) – practical development

USEFUL RESOURCES/GUIDANCE:

- http://www.ocr.org.uk/qualifications/as-a-level-gce/as-a-level-gce-computer-science-h046-h446-from-2015/
- http://www.codecademy.com/
- https://www.python.org/
- http://www.teach-ict.com/
- https://student.craigndave.org/a-level-videos
- https://isaaccomputerscience.org/home
- https://www.youtube.com/playlist?list=PL8dPuuaLjXtNlUrzyH5r6jN9ulIgZBpdo

Books

- OCR A Level Computer Science by Sean O'Byrne, George Rouse, Jason Pitt, published by Hodder Education (ISBN: 9781471839764)
- OCR AS and A Level Computer Science by PM Heathcote and RSU Heathcote (ISBN : 979 1 910523 05 6)

Curriculum & Assessment Map

A LEVEL COMPUTER SCIENCE						
YEAR 13	**AUTUMN 1**	**AUTUMN 2**	**SPRING 1**	**SPRING 2**	**SUMMER 1**	**SUMMER 2**
CONTENT	**Component 3.1:** Analysis of the problem **Component 1.4.1:** Data Types **Component 1.3.2:** Databases **Component 1.2.4:** Types of Programming Language. **Component 1.2.2:** Applications Generation **Component 1.3.3:** Boolean Algebra **Component 3.3.1:** Iterative development process	**Component 3.1:** Analysis of the problem **Component 3.2.1, 3.2.2:** Decompose the problem, Describe the solution **Component 2.3.1:** Algorithms **Component 1.4.2:** Data Structures **Component 3.3.1:** Iterative development process **Component 3.3.2:** Testing to inform development	**Component 2.3.1:** Algorithms **Component 3.2.2, 3.2.3, 3.3.2, 3.4.2, 3.4.4:** Project development Describe the solution, Describe the approach to testing, Testing to inform development, Success of the solution, Maintenance and development **Component 1.4.2:** Data Structures	**Component 2.3.1:** Algorithms **Component 2.1.2:** Thinking ahead **Component 1.4.2:** Data Structures **Component 1.3.4:** Web Technologies **Component 1.3.2:** Networks	**Component 2.1.5:** Thinking concurrently **Component 2.2.1:** Programming techniques **Component 2.2.2:** Computational methods **Component 1.3.1:** Compression, Encryption and Hashing	Study Leave
SKILLS	Develop knowledge and understanding of: -Representation and normalisation of floating point numbers in binary. -Floating point arithmetic, positive and negative numbers, addition and subtraction. -Bitwise manipulation and masks: shifts, combining with AND, OR, and XOR. -How character sets (ASCII and UNICODE) are used to represent text. -Normalisation of databases to 3NF -Reading, writing and interpreting SQL -Referential integrity -Transaction processing -ACID Analysis	Computing Project- Analysis - Decompose the game development problem, then explain and justify the structure of the solution. -Describe the parts of the solution using algorithms justifying how these algorithms form a complete solution to the problem. Develop knowledge and understanding of: -The suitability of different algorithms for a given task and data set, in terms of execution time and space. - Structures to store data: linked-list, graph (directed and undirected).	Develop knowledge and understanding of: -The nature, benefits and drawbacks of caching. -The suitability of different algorithms for a given task and data set, in terms of execution time and space. -Structures to store data: stack, queue, tree, binary search tree, hash table. Complete the following for the game project: -Describe usability features to be included in the Solution. -Identify key variables / data structures / classes justifying choices and any necessary validation. Identify the	Develop knowledge and understanding of: -Measures and methods to determine the efficiency of different algorithms -Big O notation -Comparison of the complexity of algorithms. -Standard algorithms (merge sort, quick sort, Dijkstra's shortest path algorithm, A* algorithm) -The nature, benefits and drawbacks of caching. -How to create, traverse, add data to and remove data from the data structures mentioned above. -Search engine indexing. PageRank algorithm. -Server and client side processing.	Develop knowledge and understanding of: - Determine the parts of a problem that can be tackled at the same time. -Outline the benefits and trade offs that might result from concurrent processing in a particular situation. -Recursion, how it can be used and compares to an iterative approach. - Features that make a problem solvable by computational methods. -Problem recognition. -Problem decomposition. -Use of divide and conquer. -Use of abstraction. -Apply knowledge of: • backtracking • data mining	Study Leave

	-Define the stakeholder for the problem. -Define the user requirement and stake holders. Develop knowledge and understanding of: -Procedural languages. -Assembly language -Modes of addressing memory -Stages of compilation -Linkers, Loaders and the use of libraries -The use of pipelining in a processor to improve efficiency GPUs and their uses - Use of rules to derive or simplify statements in Boolean algebra: De Morgan's Laws, distribution, association, commutation, double negation. - The logic associated with D type flip flops, half and full adders. Practical work Implementing code in Monkey-x to develop Computing project (game). Progress recorded in report.	Practical work Implementing code in Monkey-x to develop Computing project (game) -Obtain feedback from the stakeholders -Analysis of suggestions and make improvement on the coding. -Create test plan and record evidence of tests.	test data to be used during the iterative development and post development phases and justify the choice of this test data. - Use the test evidence from the development and post development process to evaluate the solution against the success criteria from the analysis. -Provide annotated evidence of the usability features from the design, commenting on their effectiveness. -Discuss the maintainability of the solution. -Discuss potential further development of the solution.	-Network security and threats, use of firewalls, proxies and encryption. -Network hardware.	• heuristics • performance modelling • pipelining • visualisation to solve problems. Develop knowledge and understanding of: -Object oriented techniques. -Run length encoding and dictionary coding for lossless compression -Symmetric and asymmetric encryption. -Different uses of hashing.	
ASSESSMENT	September internal exam Computing Project (Component 3) – Analysis	November internal exam Computing Project (Component 3) – Implementation	February internal exam Computing Project (Component 3) – Design and Testing	Submission of Computing coursework (Component 3 – 20%) Mock Exams	Mock exams	Study Leave
USEFUL RESOURCES/GUIDANCE: • http://www.ocr.org.uk/qualifications/as-a-level-gce/as-a-level-gce-computer-science-h046-h446-from-2015/ • http://www.codecademy.com/						

- https://www.python.org/
- http://www.teach-ict.com/
- https://student.craigndave.org/a-level-videos
- https://isaaccomputerscience.org/home
- https://www.youtube.com/playlist?list=PL8dPuuaLjXtNlUrzyH5r6jN9ulIgZBpdo

Books

- OCR A Level Computer Science by Sean O'Byrne, George Rouse, Jason Pitt, published by Hodder Education (ISBN: 9781471839764)
- OCR AS and A Level Computer Science by PM Heathcote and RSU Heathcote (ISBN : 979 1 910523 05 6)