Computing Programme of Study

Guidance notes for Primary and Infant schools

This is an adaptation of the original guidance notes produced by BCS, the Chartered Institute for IT, and Royal Academy of Engineering in November 2012. These notes give additional guidance on some of the words and phrases in the Draft Programme of Study (POS). Although originally written by the working group that drafted the POS these notes are not endorsed by DfE. Indeed they should be regarded as just one resource in the rich menu of online resources that support teaching and learning in computing.

Teachers who are given the responsibility for teaching any aspect of the new Computing POS are strongly advised to join [Computing at School](http://community.computingatschool.org.uk/) (CAS) and to make use of the wealth of teaching resources on the CAS and the [Primary Computing Initial Teacher Training](https://sites.google.com/site/primaryictitt/home) websites. Both sites are free of charge as are the resources they contain. Extensive use of these sites should substantially reduce the need for additional CPD. For example, this link will take you to the recent [CAS Wessex Workshop](http://code-it.co.uk/cas/) resources.

There is little need for schools to invest in additional hardware and software in order to teach the new POS. Much of the work can be based on free applications (Scratch, MSW LOGO etc.) or make use of existing applications, all of which will run on existing computers. Physical computing is one element of the POS which may require some expenditure if the school does not already have control interfaces and the software to drive them (a modest outlay of about £100 will buy an interface and the software to run it).

**Terminology**

The Computing Programme of Study deliberately uses technical terms, such as “algorithm”, “abstraction”, or “data representation”, rather than more colloquial forms, to add precision and depth of meaning to a very short document.

Technical terms have precise meanings; you can look them up in Wikipedia and other literature to draw on a rich source of ideas and inspiration. They also usually describe more general ideas than their colloquial counterparts; for example, as children develop their knowledge and understanding through Key Stages 3 & 4 they will find that an “algorithm” can be parallel or distributed, something not encompassed by “a sequence of steps”.

These guidance notes describe the meanings of some of these terms as they apply to Key Stages 1 & 2 and links to examples on the [Primary Computing Initial Teacher Training](https://sites.google.com/site/primaryictitt/home) website are provided. See Appendix B for an illustration of progression in the acquisition of computing constructs from Yr 1 to Yr 6.

[**Computational thinking**](https://sites.google.com/site/primaryictitt/home/computational-thinking) **and** [**abstraction**](https://sites.google.com/site/primaryictitt/home/key-stage-2/abstraction)

**Computational thinking** in simple terms is thinking in a logical, sequenced way to develop a solution to a problem. It is the process of *recognising* aspects of computation in the world that surrounds us, and *applying* tools and techniques from computing to understand and reason about both natural and artificial systems and processes. It is an approach to solving problems that occur not only in writing programs but also in dealing with problems in the physical world. For example, decomposing complex problems into simpler steps and solving these steps one at a time is sequencing, providing alternative solutions to solve different aspects of a problem is selection and solving problems incrementally by taking repeated actions is repetition. Sequencing, selection and repetition are part of the basic toolkit of computer programming but they are also techniques that apply to organising a multitude of common tasks such as planning a journey or decorating a house.

Computational thinking is something that *people* do (rather than computers), and includes the ability to think logically and algorithmically. A key property of computational thinking is that it must be applied to the creation of computer programs. Through the program a computer dramatically extends the scale, speed, and “reach” of what can be achieved. Within computational thinking, abstraction is a form of simplification, the hiding of unwanted or not needed information and giving only the relevant information. Due to the relative simplicity of the programs written by Keys Stage 2 children, abstraction is rarely required. For example, when planning a picnic, to ensure that everything is included; the tasks could be listed as, container, cloth, drink, bread, filling, fruit, utensils etc.. At this stage we do not need to consider what type of container is to be used to hold the picnic, the colour of cloth, white or brown bread, ham or jam etc.. These decisions can come later.

The London Underground map is a simple model of a complex reality — but it is a model that contains precisely the information necessary to plan a route from one station to another.

A procedure to compute square roots hides a complicated implementation (iterative approximation to the root, handling special cases) behind a simple interface (give me a number and I will return its square root).

Computational thinking values elegance, simplicity, and modularity over ad-hoc complexity.

[**Modelling**](https://sites.google.com/site/primaryictitt/home/key-stage-2/simulation)

Modelling is the process of developing a representation of a real world issue, system, or situation, that captures the aspects of the situation that are important for a particular purpose, while omitting everything else. *Examples: London Underground map; storyboards for animations; a web ”site map”; the position, mass, and velocity of planets orbiting one another.*

Different purposes need different models. *Example: a geographical map of the Underground is more appropriate for estimating travel times than the well-known topological Underground map; a network of nodes and edges can be represented as a diagram, or as a table of numbers.*

A particular situation may need more than one model. *Example: a web page has a structural model (headings, lists, paragraphs), and a style model (how a heading is displayed, how lists are displayed). A browser combines information from both models as it renders the web page.*

[**Decomposing**](https://sites.google.com/site/primaryictitt/home/key-stage-2/decomposition)

A problem can often be solved by decomposing it into sub-problems, solving them, and combining these solutions together to create a complete solution to the original problem. For example “Make breakfast” can be broken down into “Make toast; make tea; boil egg”. Each of these in turn can be decomposed into an even simpler set of tasks. Programming languages allow the programmer to build a relatively complex program from a number of simpler *sub-programs* or procedures. Real programs almost invariably consist of layer upon layer of procedures, each using the services of the layer below, and hiding complexity from the layer above.

The organisation of data can also be decomposed. For example, the data representing the population of a country can be decomposed into entities such as individuals, occupations, places of residence, etc.

Sometimes this top-down approach is the way in which the solution is *developed*; but it can also be a helpful way of *understanding* a solution regardless how it was developed in the first place.

[**Generalising**](https://sites.google.com/site/primaryictitt/home/key-stage-2/generalising)

Complexity is often avoided by generalising specific examples, to make explicit what is shared between the examples and what is different about them. For example, having written a LOGO procedure to draw a square of size 3 and another to draw a square of size 5, one might generalise to a procedure to draw a square of any size N, and call that procedure with parameters 3 and 5 respectively. In this way much of the code used in different programs can be written once, debugged once, documented once, and (most important) understood once.

Generalisation is the process of recognising these common patterns, and using them to control complexity by sharing common features.

[**Algorithms**](https://sites.google.com/site/primaryictitt/home/key-stage-1/algorithms)

An algorithm is a precise method of solving a problem. Algorithms range from the simple (such as instructions for changing a wheel on a car) to the ingenious (such as route-finding), and cover many different application areas (for example, making a sandwich, getting dressed, walking to school, controlling a lighthouse and so on).

An algorithm can be expressed as a program in many different programming languages.

There may be more than one algorithm to solve a single problem, differing in their simplicity, efficiency, or generality. For example, to find a path through a maze, one (simple, slow) algorithm might be to simply walk around at random until you find the exit. Another (more complicated) one would involve remembering where one had been to avoid going down the same blind alley twice. Another might be to keep your left hand on the wall and walk till you find the exit (faster, but does not work on all mazes, and so less general).

**Human factors**

It is important to remember that it is people who use information technology (IT). Children need to be aware that when designing new IT systems human factors also need to be taken into consideration. Human computer interaction (HCI) includes rules for good system design e.g. having an undo button, checking that it is clear to the user what they need to do at every stage, considering the needs of disabled users etc. These rules can be explored by examining existing IT systems (such as the Wi or the Xbox) and getting the children to evaluate their usability, i.e. what is easy to use and what is more difficult and why. They should learn to apply these rules to their own designs.

Children should also be aware of the need to understand the wider social context surrounding developing and deploying IT systems, this can be explained with reference to case studies of systems which failed to deliver the benefits hoped for because of the lack of prior consideration of the disruptive changes then system would bring to the way people work or by exploring how for example social media e.g. twitter and Facebook has change the way people interact with one another both positively and negatively.

**Socioeconomic Factors**

**Digital Divide**

It is important to help pupils realise that access to technology can bring benefits and power, but that not everyone has easy access. Lack of access to technologies can disadvantage particular groups or individuals within societies. Exclusive access to data and/or technologies can give advantage to organisations or individuals.

It is worth noting here that computational thinking and many of the fundamental principles of computing can be taught “**unplugged**”. Unplugged is computing without a computer. Typically this is computing taught through storytelling, role play, games etc..

**Gender and inclusion**

It is important to counter the stereotypes often associated with information technology and computing, e.g. that it is a male-only field. Efforts should be made in, for example, the selection of historical or contemporary case studies to reflect the positive contributions of female practitioners, for example Ada Lovelace, Grace Hopper or Dame Wendy Hall.

Projects topics should also be carefully considered to be inclusive to both genders.

**Assistive technology**

As with other areas of the curriculum, IT can be made more accessible to children with some special educational needs or disabilities through the use of assistive technology, from adapted mice or keyboards to screen readers and Braille displays. Within the curriculum, pupils might evaluate whether digital content is accessible to users with SEND, and learn about assistive technology as examples of ‘forms of input and output’ at KS2 and ‘hardware and software components’ at KS3.

**English as an Additional Language**

Technology can also facilitate the inclusion of children learning English as an additional language. The user interface of the operating system or application software can be set to languages other than English, and, for example, Scratch programs can be written in languages other than English. LOGO requires a very limited vocabulary of words that can be quickly learnt by very young children.

Machine translation technology allows instructions, questions and responses to be translated automatically, often with a good degree of accuracy between common languages; teachers may wish to explore the process and accuracy of such services. Machine translation may also be useful for project work in which pupils learn about the opportunities offered by the Internet.

**Purpose of study**

A high-quality computing education equips pupils to understand and change the world through computational thinking. It develops and requires logical thinking and precision. It combines creativity with rigour: pupils apply underlying principles to understand real-world systems, and to create purposeful and usable artefacts. More broadly, it provides a lens through which to understand both natural and artificial systems, and has substantial links with the teaching of mathematics, science, and design and technology.

At the core of computing is the science and engineering discipline of computer science, in which pupils are taught how digital systems work, how they are designed and programmed, and the fundamental principles of information and computation. Building on this core, computing equips pupils to apply information technology to create products and solutions. A computing education also ensures that pupils become digitally literate – able to use, and express themselves through, information and communication technology – at a level suitable for the future workplace and as active participants in a digital world.

**Aims**

The National Curriculum for computing aims to ensure that all pupils:

* can understand and apply the fundamental principles of computer science, including logic, algorithms, data representation, and communication
* can analyse problems in computational terms, and have repeated practical
* experience of writing computer programs in order to solve such problems
* can evaluate and apply information technology, including new or unfamiliar
* technologies, analytically to solve problems
* are responsible, competent, confident and creative users of information and
* communication technology.

**Attainment targets**

By the end of each key stage, pupils are expected to know, apply and understand the matters, skills and processes specified in the relevant programme of study.

**“Digital literacy, information technology, and computer science”.** Each of these elements should be visibly present at every stage in a pupil’s IT education. However, the three are closely related, overlap, and should not be thought of as “silos” into which lessons can be categorised.

For example, use of a spread sheet is using information technology, knowing when to use a spread sheet is an example of digital literacy, but use of formulae to add up data or find an average crosses over into an exercise in programming (computer science).

To take another example, the PoS asks that pupils are taught about how search engines rank searches (computer science). The context is likely to be about finding relevant information to help with a problem; clarity about the problem can make them better at searching (information technology). The link can then be made to the wider context of the need for efficiency in finding information, assessing its reliability and using it responsibly; avoiding breach of copyright and plagiarism (digital literacy).

**When planning the implementation of a computing curriculum it is important not to abandon the excellent information technology activities already embedded in the wider curriculum to fulfil the old, ICT POS. Opportunities should be identified to enhance the children’s use of information technology across the curriculum through the inclusion of computing activities. In this way schools should have little difficulty delivering the new Computing POS in their entirety. An improvement of the new Computing POS is the emphasis that it places on intellectual rigour over content. See Appendix C for a comparison of ICT and Computing.**

**“Societal value”.** This term invites teachers to help the children to reflect on some of the effects that pervasive information technology has on the society in which we live. At Key Stage 1 this will be introduced to these ideas through working collaboratively using IT and some age appropriate E safety teaching.

During Key Stage 2 children will gradually be introduced to communications technology and social networks via email, on line games, virtual learning environments, blogs etc.. Such activities will provide ample opportunity for teachers to discuss with the children the impact of IT on society. Close links should be made to E safety provision which, as a safeguarding children issue, should extend beyond the Computing curriculum and include an element of parent education.

**“Create purposeful and usable artefacts”** means that there should be a reason, clearly identified to the children, or by the children themselves, why a computer system or artefact is created. A system can be thought of as *a computer and some software to make it work*. An artefact can be thought of as *anything created on and/or stored on a computer*. It might be a picture, music, a video, a game, a graph, a story etc. This is not new. It has always been good practice when teaching ICT to discuss with the children when its use is helpful and appropriate and when it is not. The ability to be discerning about their use of ICT is an important aspect of a child’s digital literacy. It is important that learning should be set within a context that the children readily understand. Ideally this should come from the wider curriculum.

**Key stage 1**

**Work individually and collaboratively”.** A fundamental aspect of computing and information technology in the workplace and beyond is that it is collaborative. Systems are invariably built by teams; pair-programming and peer code review is well-established professional practice; developing software components that can be used by others is the key to modularity; and so on.

**It is not appropriate to try to *teach* professional software engineering practice at school, but it is important that from the earliest age children should have the *experience* of describing their digital creations to others, and working together to develop, critique, and improve them. Children should not consider that they have solved a** [**programming**](https://sites.google.com/site/primaryictitt/home/key-stage-1/write-and-test-simple-programs) **problem until they have invited their peers to test their solution.**

“**Work creatively**”. Unlike natural science where we discover facts about the natural world, computer science and information technology are entirely the result of human creativity. It is through the creative processes of making and refining programs and digital media that children acquire for themselves a deep understanding of how technology works and the principles that underpin it. In IT, as in other creative disciplines, theoretical understanding is developed in parallel with increasing practical capabilities.

Valuing creativity also allows for computing and IT to be taught in collaboration or sympathy with other, arts subjects, creating a digital basis for personal expression, as well as further academic progression or work in the digital or creative economy (e.g. games, graphics, video, animation, and interactive technology).

“**Playfully**”. A playful attitude towards technology motivates and encourages children to develop independence, confidence and understanding, and sits in a long tradition including Froebel, Papert and, more recently, the work of MIT’s Lifelong Kindergarten Group. An attitude of playful experimentation and exploration characterizes the work of many software developers and computer scientists.

“**A range of devices”** here indicates digital devices accepting input, producing output and operating according to a stored program, including desktop computers, mobile phones, digital cameras and game consoles; the operation of these devices are controlled by computers. Programmable devices are those where the user themselves can create or alter the program, such as a desktop or laptop computer, smart phone or tablet. A device might be a model built in a DT lesson, such a lighthouse or a fairground ride that the children control via a computer and interface ([physical computing](https://sites.google.com/site/primaryictitt/home/key-stage-2/control)).

**“Algorithm”.** At KS1 an algorithm is likely to be no more than a simple sequence of steps (e.g. open bread bin; cut slice; put bread in toaster; wait; take toast out; eat it). Sandaig Primary – Romy Robot is a good starting point <http://www.sandaigprimary.co.uk/fun/rommy_robot.html>. This is an algorithm to post a letter narrated by a seven year old - *‘I need to go to the end of my road then turn left at the High street. On the High street I need to go to Tesco to buy a stamp. Then I go to the end of the High street and turn right. A bit along Station Rd there is a letter box. I stop and post the letter’.*

**“Simple programs”.** These may be sequences of instructions for [controlling](https://sites.google.com/site/primaryictitt/home/key-stage-2/control) the movement of a robot (eg Bee Bot, ProBot or Big Trak programmable toy) or an on screen turtle or sprite. Bee Bots have been in schools for many years now. Some Early Years centres even have them. The algorithm can be thought of as the *plan* for a program. The *program* is the algorithm written using a *code or language* that the computer can understand and follow.

**“Organise, store, manipulate, and retrieve data”** includes the efficient and effective use of the computer file system or equivalent cloud-based storage.

**Key stage 2**

**“Collecting, analysing, evaluating and presenting data and information”,** should make use of the specific capabilities of IT to extend these activities through interactivity, automation and increased speed, capacity (e.g. using large data sets) and range, as well as through joint projects mediated via the Internet. Information is understood here as data to which a specific meaning has been attached through a process of interpretation.

For example in a STEM project the children are studying ergonomic design and the skeleton. They want to build up a ‘picture’ of an average child so they take measurements form their own bodies and record the data for the entire year group in a spread sheet. The data is then analysed, graphs are produced and used to inform the design process.

**In** [**programs**](https://sites.google.com/site/primaryictitt/home/key-stage-1/programs)

* **Sequence** means putting instructions in order to be executed one after another.
* In a [**selection**](https://sites.google.com/site/primaryictitt/home/key-stage-2/selection)structure, a question is asked, and depending on the answer, the program will choose between two or more possible courses of action. At KS2, selection should include the if..then..else statement. (E.g. **If** the sprite is touching a wall **then** bounce back, **else** move forward.).
* [**Repetition**](https://sites.google.com/site/primaryictitt/home/key-stage-2/repetition)means repeating a sequence of instructions a certain number of times, or until some specific result is achieved. In programming terms this means loops of all kinds, such as repeat, for, while, until etc. (e.g. move dog 1 step forward; repeat until dog is in kennel then stop).

**“Various forms of input and output”.** Keyboard, mouse, sensors, screen, speakers, microphone, temperature sensor, fairground roundabout.

**“Understand** [**computer networks**](https://sites.google.com/site/primaryictitt/home/key-stage-2/computer-networks)**”** means, at this stage, knowing that a network consists of one or more computing devices connected together, using shared protocols, so they can share data and resources. “**Internet services**” means things like school blogs, web-based spread sheets, language translation services, social networking sites or email.

**“Opportunities for communication and collaboration”** is one of the most immediate and visceral impacts of the internet on pupils’ lives. Children should personally experience opportunities to communicate and collaborate both internally within the school and, where possible, externally. That experience should in turn inform, and be informed by, reflective discussion about issues such as respectful communication in a context where body language is absent; cultural differences; privacy; and safety.

**“How they change over time”.** As well as thinking about how technology has evolved during their parents’ or grandparents’ life times, children should be taught about some of the key figures and events in the development of IT, many of them from the UK, including

* Charles Babbage, Ada Lovelace and the difference and analytical engine
* Alan Turing, the Turing Test, Turing machines, Enigma and the work of Bletchley Park
* Tim Berners-Lee and the invention of the Web.

Children should also consider some implications of the continuing technological innovation in their and others’ lives, perhaps creating digital content to illustrate how they think technology may further change over the next ten or twenty years.

**“Appreciate how [search] results are selected and ranked”**. [Internet](https://sites.google.com/site/primaryictitt/system/app/pages/search?scope=search-site&q=internet) search engines must choose which order to present results in. The “page rank” algorithms used to do so are interesting in their own right, and elementary versions are accessible even at KS2 (e.g. pages to which many other pages point are more highly ranked). Given the enormous influence of search engines, other non-technical factors come into play, notably advertising and censorship.

**Appendix A - Recommended CPD Resources:**

Access this document on line at - <https://sites.google.com/site/primaryictitt/home>

|  |
| --- |
| (click on the link to **Additional content**)   * [Opening Keynote](http://code-it.co.uk/cas/jwoollardag.pptx) CAS Wessex conference (John Woollard) * [An excellent overview video](http://www.youtube.com/watch?v=dZ3pQhcMm0M) – first 7 mins (Southampton University) |
| * [Computer Science v ICT Explained](http://www.resources.digitalschoolhouse.org.uk/recommended-reading/174-dsh-recommended-computing) (Mark Dorling – Digital Schoolhouse) * [Computational Thinking](http://www.iste.org/learn/computational-thinking) + [Progression chart – curriculum mapping tool](http://www.iste.org/docs/ct-documents/ct-teacher-resources_2ed-pdf.pdf?sfvrsn=2) (iste.org) * [Evolving ICT into Computing Primary KS2](http://code-it.co.uk/cas/Farewell%20to%20ICT%20or%20evolving%20ICT%20into%20Computing.pptx) (Phil Bagge) |
| * [KS1 Primary Briefing](http://code-it.co.uk/cas/KS1%20Briefing.pptx) (Emma Goto) |
| * [Code-it.co.uk KS2 Computer Science Planning](http://code-it.co.uk/csplanning.html) (Phil Bagge) |
| * [Fun Computing Topics](http://code-it.co.uk/cas/Fun%20Computing%20Topics.pptx) (Dan Gardner) * [Thinking Myself](http://games.thinkingmyself.com/) (a computational thinking game for children and beginners) |
| * [Control Technology Progression KS1-3](http://code-it.co.uk/cas/KS1_to_3_Control_Technology_-_Progression_%28Year1_to_Year_8%29.xls)(Graham Hastings) |
| * [Primary Ninja Turtles Logo in KS2](http://code-it.co.uk/cas/Primary%20Ninja%20Turtles.pptx) (Phil Bagge) |
| * [Scratch That Scratch in KS2](http://code-it.co.uk/cas/Scratch%20That.pptx) (Phil Bagge) * Using [Scratch](http://scratch.mit.edu/educators/) to teach Computer Science at Key Stage 2 via the [Raspberry Pi](http://store.raspberrypi.com/projects/casmanual)   [Education Manual](http://store.raspberrypi.com/projects/casmanual) (you do not need a Raspberry Pi computer - any pc will do) |
| * [1010 Reasons for Teaching Computer Science](http://www.youtube.com/watch?v=OCtD-O8D-mk) (An overview by Miles Berry) |
| * [How the Internet Works](http://www.slideshare.net/lescarr/what-istheinternet-17716099) (Les Carr) |
| * [KS1 Bee Bot Workshop](http://code-it.co.uk/cas/KS1BeebotWorkshop.pptx) (Emma Goto) * [Romy Robot](http://www.sandaigprimary.co.uk/fun/rommy_robot.html.) – algorithms and programs (Sandaig Primary) |
| * [KS2 Logo Session Video 40mins](http://archive.zepler.tv/467/) (Phil Bagge) |
| * [KS2 Scratch Session Video 35mins](http://archive.zepler.tv/469/) (Phil Bagge) * [Jam sandwich algorithm](http://code-it.co.uk/resources/sandwich_algorithm.pdf) (Phil Bagge) ([out takes video](http://youtu.be/leBEFaVHllE)) * [Physical Computing](http://youtu.be/KNFzh3M0fJ4) (Graham Hastings) * [Overview Prezzi](http://prezi.com/bn69lfspag0h/present/?auth_key=a5qt3j0&follow=qsxgqwbrfxfy) (Graham Hastings) * [Access this document from the CAS web site](http://community.computingatschool.org.uk/resources/791) (Graham Hastings) |

**Appendix B – Progression in computing as a series of ‘can I?’ questions:**

There is a good deal of repetition as the children will revisit the computing constructs in a variety of different contexts and with an increasing degree of sophistication.

Can I statements…

**Year 1**

Can I explain what an algorithm is?

Can I write a series of instructions?

Can I test if my instructions work?

**Year 2**

Can I explain what I think the program will do?

Can I write, test and improve a program?

**Year 3**

Can I write a program that will perform a set goal?

Can I explain how an algorithm works? (part of a program)

Can I use repetition in my programs? (loops)

**Year 4**

Can I write a program that will perform a set goal?

Can I explain what function an algorithm will perform?

Can I debug my work? (Can I spot errors, correct and improve my work?)

Can I use repetition in my programs? (loops)

Can I use variables in my programmes? (such as names, scores or levels)

**Year 5**

Can I write a program that will perform a set goal?

Can I solve a problem by breaking it into smaller parts?

Can I write a program to control a physical system or simulated physical system?

Can I debug my work? (Can I spot errors, correct and improve my work?)

Can I use variables in my programmes? (such as names, scores or levels)

Can I use conditionals in my programs? (if, then, else)

Can I use inputs and outputs?

**Year 6**

Can I write a program that will solve a problem?

Can I solve a problem by breaking it into smaller parts?

Can I write a program to control a physical system or simulated physical system?

Can I debug my work? (Can I spot errors, correct and improve my work?)

Can I use conditionals in my programs? (if, then , else)

Can I use variables in my programs?

Can I use physical inputs and outputs?

With thanks to CAS member [**Nicholas Hughes**](http://community.computingatschool.org.uk/users/4473)

**Appendix C – Computing v ICT Explained**

|  |
| --- |
|  |

|  |
| --- |
| **Why computing?**  Originally the DSH lessons were focused on the teaching of ICT. The experience in the DSH confirms that KS2 pupils are often far more capable than the existing ICT curriculum allows for. Although functional ICT skills are important, if we expect pupils to become independent learners, evaluators and designers of new technologies then it is important that they have a clear understanding of the computing principles and concepts that underpin these technologies.    **How is Computer Science different to ICT?**   * ICT can be compared to driving a car. * Computer Science can be compared to designing a car.   **Computing is a STEM subject**  It is often said that Computer Science is the silent ‘C’ in STEM (Science, Technology, Engineering and Maths) and the DSH believes that Computer Science should be treated on equal status with the other STEM subjects.    The DSH believes that actually there is a second and ‘more’ silent ‘C’ in STEM, it is Creativity. Creativity is going to be equally as important to pupils in their future use of, and designing of, new technologies. This combination of Creativity and Computing is going to be critical to the UK economy in the 21st Century!    **Computing without computers**  The DSH accelerated learning model enables the teaching of KS3 and KS4 concepts to key stage 2 pupils. This is achieved by challenging their perceptions and their expectations when working in an ICT suite, with Computer Science concepts being taught (wherever possible) without the use of computers and related to pupils existing ‘real world’ understanding. If this approach to teaching Computer Science interests you, The Digital Schoolhouse would recommend that you visit the [CS Unplugged website](http://csunplugged.org/activities) that contain lots of clear, fun and free resources to download. |

With thanks to Mark Dorling and the [Digital Schoolhouse](http://www.digitalschoolhouse.org.uk/)