Lower Key Stage 2 – *Programming and Games* – Theme Guide

Children explore simulations, investigating the structure and exploring how they might be programmed. They begin to note that abstraction can simplify them. They decompose tasks, creating and debugging algorithms and understanding how algorithms support the programming process. They write, test, debug and refine programs to achieve specific objectives, using sequence, repetition and procedures. They explore selection in digital systems.

Learning objectives for the term
To understand a simulation is a digital system with specific rules, providing an environment often mimicking real world situations.
To understand abstraction leaves out unnecessary detail and is helpful when designing a process. *Begin to see that abstraction can help us identify similarities between processes*.
To understand simulations are programmed to allow choices by the user to change the outcomes.
To understand we can use algorithms to design the steps of a process before writing computer programs.
To understand a program is a sequence written in a programming language and designed to perform a specific task.
To know problems can be solved efficiently using decomposition and that this is central to good programming practice.
To understand program commands can be saved as a procedure and procedures can be called by programs and procedures.
To develop independent programming capability.
To understand selection is a programming process, which uses a yes/no question to provide alternative routes through a program.
To be aware that online simulations may include chat facilities and to know how to stay safe around these.
On-going Learning Objectives
To review and evaluate their work, discussing the choices they have made and checking for accuracy.
To use appropriate file-name conventions and understandable folder structure to save, organise and retrieve their work.
To understand essential eSafety rules and to know what to do in the event of an incident or concern at home or school.

Vocabulary – see Glossary for definitions (for terms in blue)	
simulation, *logical reasoning,* *algorithm,* *abstraction,* *selection,*	*program* (noun and verb), *refine,* *procedure,* *tinkering*

Possible resources for this theme (further resources are suggested with the explanatory notes below. Note that these are examples and not formal recommendations.)	
Programming Apps • ScratchJR (iPad / Android) • Logotacular (iPad) • Swift Playgrounds (iPad) • Pyonkee (iPad) • Tynker (iPad / Android) • Lego® Mindstorms® Fix the Factory (iPad / Android) **Free Online programming tools** (read the terms and conditions / privacy policy before using.) • Rapid Router: https://www.codeforlife.education/rapidrouter/ • Scratch: https://scratch.mit.edu/ • Code.org: https://code.org/	**Programming Software** • Scratch • 2Code (as part of 2Simple Purple Mash) • Espresso Coding • Kodu Game Lab **Programmable devices** • Sphero / Ollie • Roamer Too • Bluebot (with accompanying app) **Online Simulations** • See accompanying sheet [*Links to Simulations*.]

Please note that with any online platform it is essential that you review the privacy policy and terms and conditions of the service. The school is responsible for the protection of data it holds and compliance with current data protection legislation. Always assess both the data protection and safety of the service you are considering using, and ensure the necessary permissions are in place before using with pupils.

Free *Barefoot Computing* activities to support this theme.	
• **Solar System Simulation Activity.** Link Pupils use Scratch to program a solar system simulation. • **2D Shape Drawing Debugging Activity.** Link Children debug algorithms used for drawing 2D shapes. • **Logical Number Sequences.** Link Children work with number sequences to learn about rule-based algorithms. • **Decomposition Unplugged Activity**. Link Children use hand-clapping sequences, breaking them down into individual steps (decomposition.) • **ScratchJR (app) Knock-knock Joke Activity.** Link Children program an animated knock-knock joke using the ScratchJR app. • **Scratch Tinkering Activity**. Link Pupils explore Scratch and learn about how it works. • **Fossil Formation Animation Activity**. Link Pupils use Scratch to program an animation that illustrates the steps in fossil formation. • **Viking Raid Animation**. Link Pupils use Scratch to program an animation of a Viking raid. • **Abstraction Unplugged Activity**. Link Children create drawings and/or models and guess what their peers are creating. • **Make a Game Project**. Link Children program a game using Scratch. • **Kodu Game Selection Activity**. Link Children program a game using Kodu Game Lab. • **Scratch Maths Quiz Selection Activity**. Link Children create a maths quiz in Scratch, using selection to control the flow.	*Barefoot Computing* provides freely available resources to support teachers in delivering the computer science aspects of the 2014 Primary National Curriculum for Computing. Free registration with the *Barefoot Computing* website is required to view and download these resources. To register, visit: http://barefootcas.org.uk References to these resources and the accompanying links are provided with permission from *Barefoot Computing*.

Primary Computing Scheme online materials that are referenced in this guide can be accessed from: http://www.hertsforlearning.co.uk/user/login

You will need to be logged into your school account and have a current subscription to the Primary Computing Scheme to gain access. The materials can be accessed from the *My Resources* link at the top/right of the screen, once you are logged in.

Key learning objectives (some objectives might be used for more than one lesson)
To understand that a computer simulation is a digital environment with specific rules, which often mimics real world situations.
• Talk to the children about situations where computers can make a representation of the real world. Where do we see this? Sometimes the 'world' is a fictional one, but it is based on our real world (e.g. in many games.) • How can this be helpful to us? Why do we have simulations? (For example, why do pilots of passenger planes train on simulators and not on real planes with real passengers?) What are the limitations? What can you not simulate (or is harder to simulate) through a computer? • See attached sheet of links for some real world examples of simulation being used: [*Links to videos of 3D modelling*.] • Carry out an activity where children gather and present information about where simulations are used in the world around us, from games to training, planning (e.g. roads and traffic control) and design of buildings etc.

To understand abstraction leaves out unnecessary detail and is helpful when designing a process. *Begin to see that abstraction can help us identify similarities between processes*.
• Abstraction is something we do all the time, especially in school. We leave out information and details that are unnecessary to us. We simplify things to make them more understandable, concentrating only on the key information that is important to us. Choosing what is important and what to leave out is *abstraction*. • Show how objects around us could be represented as a simple shape, for example a wheel is a circle. • You could play a game with the children where one person draws something and the other one has to guess what it is their partner is drawing. The person drawing will make a decision about what the important parts to draw are, and also simplify their details, so that their partner can guess. This is abstraction. If drawing a house, the drawer will probably simplify it to a square with a triangular roof, one door and 4 windows, rather than putting in lots of detail that will make it no easier to guess. • Children in pairs could try and represent everyday objects as combinations of simple shapes. For example a bicycle made out of triangles and circles. • Through abstraction we can begin to see that some processes which appear quite different are actually very similar. This is described further in the accompanying sheet [*Abstraction Activity*.] • See this free Barefoot Computing activity (link above):   ○ Abstraction Unplugged Activity. • In most simulations, some of the details are left out. Some simulations are far more detailed and realistic than others, depending on their purpose. • Compare two simulations. Look at how they are controlled, what the options are and what detail has been left out and what has been included. Why have these choices been made? • See the accompanying sheet for some suggestions of online simulations: [*Links to Simulations*.] • See the final objective in this learning theme about eSafety when using online simulations an online games, and make sure this message is reinforced through the term (and as an on-going message.)
To understand that simulations are programmed to allow choices by the user to change the outcomes.
• Children could create a diagram or flow chart to show how one particular, simple simulation works. Make sure the simulation chosen is very simple. For example, a hot air balloon simulation may only have one control to fire the burner (and therefore make the balloon go higher) and the balloon will descend if the burner is not fired. If you use a more complicated simulation it will be hard to create a diagram or flow chart. You could choose different simulations for different pupils, as a way of differentiating. • Children could use a computer program to make their diagrams or flow charts, for example the PureFlow app (for iPad) by Aleksandr Kozlov, or using the 'shapes' tool in Microsoft® Office. • See this free Barefoot Computing activity (link above):   ○ Solar System Simulation Activity
To understand that we can use algorithms to design the steps of a process before writing computer programs.
• Away from the computer, create algorithms for specific tasks. Children will most probably have begun writing algorithms for simple tasks in KS1, so this should be a progression from then. There are lots of tasks you can choose from for this, e.g. draw a 2D shape, put on a shoe, fill a water bottle etc. • You could use or adapt the accompanying sheets for children to write their algorithms: [*My Amazing Algorithm*.] • Make sure children have the opportunity to test their algorithms on their peers, and to refine, debug and improve them based on the outcomes of the testing. Discuss the importance of doing this. • Talk about how algorithms are used before programs are written, to plan the process that the program will follow. Why is this useful? Why not just jump straight in and start programming? • See the accompanying sheet: [*10 ideas for teaching algorithms*] • See these free Barefoot Computing activities (links above):   ○ Decomposition Unplugged Activity.   ○ 2D Shape Drawing Debugging Activity.   ○ Logical Number Sequences
To understand that a program is a sequence written in a programming language and designed to perform a specific task.
• Use unplugged programming as well as physical devices and onscreen programming languages (such as Logo and Scratch) to program tasks, such as creating 2D shapes or patterns. • Remind children about the difference between algorithms and programs. Programs are written in a specific language, using only the terms and syntax of that language, for a computer to understand. Algorithms are

written for people to understand. If we typed our algorithms into a computer programming environment, it would not work! • We can still work unplugged when teaching programming. You could, as a class, choose a task and then create your own 'programming language' for an imaginary robot to follow. To do this you will need to consider carefully the exact commands that will make up your language, and when using it only these words can be included. • When using physical programmable devices, avoid jumping straight into using them and instead pupils should plan out the program (usually a series of directional commands, depending on which device you are using) on paper and/or using a map designed for the device. See the accompanying sheet: [*Show me the way activity*.] • When using on screen programming environments, consider using more than one so that children gain experience of doing the same thing in different ways. For example, you can draw geometric patterns in both Scratch and Logo, by creating a simple shape and then rotating slightly before drawing it again. This process is repeated to draw the pattern. • Encourage children to write programs and then show them to their peers so that they can predict the outcomes of the program, before testing them. Children could work together to find bugs in each other's programs and suggest improvements. • Remind children of the Repeat command, and how it can improve efficiency. • See the video guides: [*Getting Started with Logo in Primary Schools*] • See the downloadable sheet: [*MSW Logo, The Basics*] • See the video guide: [*A programming activity in ScratchJR (iPad)*] • See the downloadable activity: [*Debugging Activity – Logo*] • See the downloadable activity: [*Debugging Activity – Scratch – Shark Screensaver*] • Any of the free resources from Barefoot Computing (links above) may be helpful for this objective.
To know that problems or tasks can be solved efficiently using decomposition and that this is central to good programming practice.
• Decomposition is the process of taking a task and breaking it down into the steps that are required to complete the task. Doing this enables us to consider each step and therefore makes the task easier to understand and carry out, rather than considering it as a whole. • For example, a recipe to bake a cake. The task is to bake the cake, but in order to achieve this we have to go through a series of steps. This is the recipe. • When writing an algorithm and/or program, we use the process of decomposition to work out the steps needed to achieve the objective of the algorithm/program. The more complex the task, the more we need decomposition to help us achieve it. • Children will be familiar with following instructions, not just in school, but frequently children use services such as Youtube® to learn how to do things at home. In these videos decomposition will have been used to break the task down and make it easier to follow. • Using a more complicated objective than in the algorithms the children created above, as a class, break down some tasks into steps, using decomposition. Each step could be individually tested and debugged as necessary. • Then, apply the same process to using a programming language such as Scratch, ScratchJR or Logo and/or with physical devices such as Probot™, Roamer® Too or Sphero®. • See this free Barefoot Computing activities (link above):   ○ Decomposition Unplugged Activity.
To understand program commands can be saved as a procedure and procedures can be called by programs and procedures.
• A 'procedure' (which may also be referred to as a 'subroutine', 'subprogram' or a 'function') takes a series of lines or commands and groups them together into one unit, which may be called upon by the program. It makes the program more efficient, as every time it requires the same series of commands it just needs to call the procedure rather than having all the commands written again. • Procedures would be especially useful in programs controlling a task that is repeated again and again. • For example, a 'robot' in a factory is carrying out the same task repeatedly, and the program it follows may contain procedures to make this more efficient and easier to program. • The programming language Logo is ideal for teaching procedures. See the accompanying sheet: [*MSW Logo, the basics*] and the Logo video guides available from the online area. • Children could take the programs they wrote in Logo above, and further refine and improve them by using procedures. For example, the commands used to create a geometric pattern from drawing and rotating a square

could be put together into a procedure called '*pattern*'. Then, we only need to write the single command '*pattern*' into the programming line to draw the pattern. • Always provide the opportunity to debug, refine and improve their programs – working in pairs is good for collaborative problem solving. • See the downloadable activity: [*Debugging Activity – Logo*]
To develop independent programming capability.
• Provide time for tinkering when children use a preferred programming language to write a program of their choice. • As above, it may be beneficial to use different programming languages so that children can experience different ways of doing similar things. The important thing is that while the language may be different, the underlying concepts are the same. • Encourage collaborative work where pupils share their programs within the class, predict each other's programs and help debug and refine them. • See these free Barefoot Computing activities (links above):   o Scratch Tinkering Activity   o Make a Game Project
To understand that selection is a programming process, which uses a yes/no question to provide alternative routes through a program.
• In selection, a question is asked by the algorithm / program and the route followed thereafter is dependent on the answer. For a simple example, put a selection of red and green toy bricks into a bag and ask a child to draw one brick out. Is the toy brick red? If yes, then put it in the red bag. If no, put it in the green bag. • As a class, create a flowchart for a simple task on the board/screen, showing how we might show choice or selection in our diagram (it will branch into two.) • To include input / output in our teaching, use an example of selection applied to a device where something happens if a condition is met. E.g. a burglar alarm uses selection depending on input, and produces an output as one of the routes followed depending on the answer to the question. The burglar alarm has a device that monitors for movement (the input.) It asks the question, "Can I detect movement?" If no, then it continues monitoring. If yes, it triggers the alarm to make a sound (the output.) • You could set up a role-play where a 'robot' responds to different conditions. E.g. children write conditions (selection) such as 'does the robot hear a pop? If yes, he jumps. If no he continues to listen'. And 'does the robot hear a hiss? If yes, he holds up his arms, if no, he continues to listen.' • As always, give children the opportunity to improve their algorithms through testing, debugging and refining. • Discuss the different between input and output in these situations. How input causes the output, because a program (or algorithm) is written to control this. • See these free Barefoot Computing activities (links above):   o Scratch Maths Quiz Selection Activity   o Make a Game Project   o Kodu Game Selection Activity
To be aware that online simulations may include chat facilities and to know how to stay safe around these.
• Continue to reinforce the key eSafety messages at every opportunity. • The majority of online games and simulations, even some educational ones, enable online contact with others, and we should always be careful with what we share online, and who we communicate with. • Discuss protecting their identity when online and not accepting requests from people we do not know and trust in the real world. • Talk about what we should do if something online concerns us. • Consider creating an algorithm or flowchart (using selection and/or a procedure) for using an online game or suchlike safely. E.g.:   o Did you receive a message or friend request? *Yes/No*.   o If *yes*, do you know and trust the person in real life?   o If *yes*, accept the request (or respond to the message.)   o If *no*, ignore the request / message

On-going Learning Objectives
To review and evaluate their work, discussing the choices they have made and checking for accuracy.
To use appropriate file-name conventions and understandable folder structure to save, organise and retrieve their work.
To understand essential eSafety rules and to know what to do in the event of an incident or concern at home or school.

Suggested independent task – any open-ended activity (2-3 sessions) enabling the children to demonstrate their computing capability around the knowledge and understanding provided in the term
➢ Design an algorithm for an onscreen programming task (for example a pattern or design), which includes repeat functions and ideally procedures. ➢ Use an onscreen programming language to write the program. ➢ Test, debug and refine their program considering how to improve its efficiency ➢ Predict the outcome of a program produced by another pupil ➢ Evaluate their completed task including noting where they used decomposition to support the design of the task Other considerations: Does the task provide for children to work at different levels? Is there support available for children to select if they wish? Are there opportunities for the children to review and develop their work? Is there an opportunity for the children to evaluate the finished task?

MSW Logo is a good tool to use for this task (assuming it has been covered during the term) as repetition and procedures are clear and simple to create. Before actually creating the program on the computer, pupils should show their understanding of the concepts by creating an algorithm, showing the steps that the program will follow (using decomposition.) Provide time for the children to test their programs and improve them before submitting the final version, aiming to make it as efficient (short) as possible to achieve the given objective.
Pupils can share their programs with each other, predicting what will happen if they are run (i.e. by reading and understanding the program before running it to see if they were correct.)
Children should write an evaluation of the task, which will ideally show their understanding of the key concepts they may have used, e.g. decomposition, procedures, repetition.
See the sample independent task in the assessment section, for this learning theme.

Please note there is an example medium term plan for this theme, donated by a Hertfordshire school, available to download from the online area.