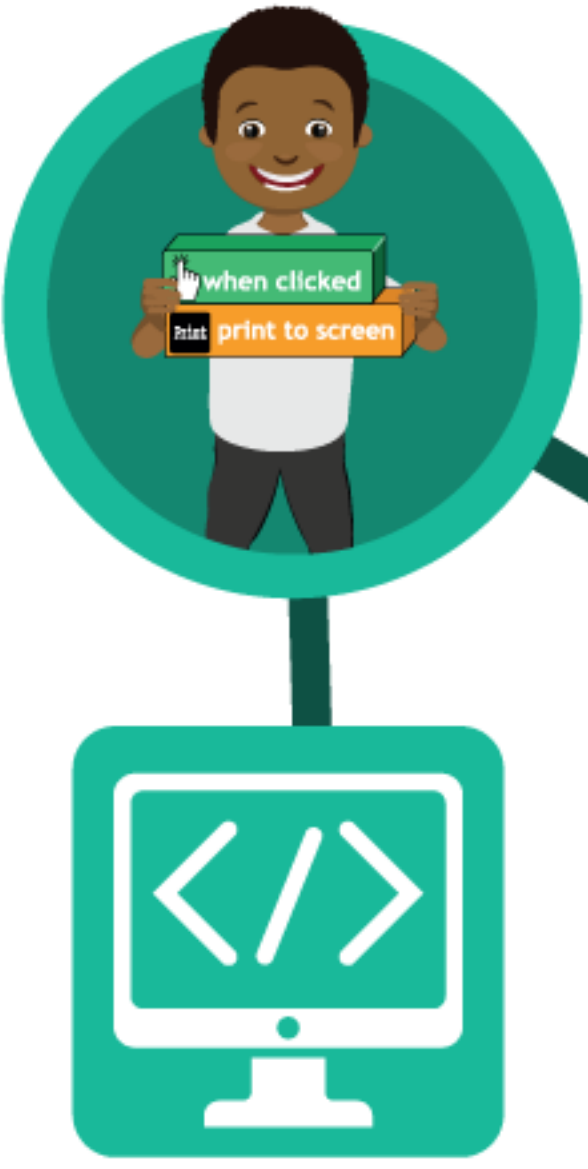
purple mash

Computing

Scheme of Work

Unit 4.1 - Coding



Year Group: 4
Number of Lessons: 6

From 2simple

Contents

Introduction

This unit consists of six lessons that assume children have followed the Coding Scheme of Work in Years 1 to 3. If most of the class have not, use the Coding Catch-Up unit instead of this unit.

New coding vocabulary is shown in **bold** within the lesson plans, use these new words in context to help children understand the meaning of them and start to build up, their vocabulary of coding words.

The Gibbon guided activities provide further practice of the concepts that the children will be learning and can be used as extension activities. More able children can be encouraged to explore other things that they can change in their programs and experiment with the options available, such as variables and 'if' statements.

Children will often be able to solve their own problems when they get stuck, either by reading through their code again or by asking their peers; this models the way that coding work is really done. More able pupils can be encouraged to support their peers, if necessary, helping them to understand but without doing the work for them.

Program Design

To master coding skills, children need to have the opportunity to explore program design and put computational thinking into practice. The lesson plans incorporate designing before coding in some lessons.

Storyboarding their ideas for programs. For example, creating a storyboard when planning a program that will retell part of a story.

- Creating annotated diagrams. For example, creating an annotated diagram to plan a journey animation that tells the story of an historical event they have been studying.
- Creating a timeline of events in the program. For example, creating a game program against the computer, what are all the actions needed from the objects?
- Using the 2Chart tool to create flowcharts of the processes.

During the design process, children should be encouraged to clarify:

- the characters (objects and their properties)
- what they will do (actions and events)
- what order things will happen (the algorithm)
- rate their confidence at being able to code the different parts of their design and either refine the design or review possible solutions as a class or group.

Levels of Scaffolded coding tasks

You can support children's learning and understanding by using different degrees of scaffolding when teaching children to code. The lessons provide many of these levels of scaffolding within them and using Free Code Chimp, Gibbon and Gorilla enables children to clarify their thinking and practice their skills. These are not progressive levels, children can benefit from all the levels of activities at whatever coding skill level they are:

Scaffolding	Task type	Examples of how to provide these opportunities
Most scaffolded ↓ Least scaffolded	Copying code	By giving children examples of code to copy.
	Targeted tasks	• Read and understand code • Remix code to achieve a particular outcome. • Debugging. • Use printed code snippets so that children can't run the code but must read it. • Include unplugged activities and 'explaining' tasks e.g. 'how do variables work?'
	Shared coding	• Sharing Challenge activities as a class or group on the whiteboard. • Complete guided activity challenges as a class. • After completing challenges; share methods to create a class version of the challenge. • Free coding as a class
	Guided exploration	• Exploring a limited repertoire of commands • Remixing code • Explore commands in free code before being taught what they do. • Use questioning to support children's learning.
	Project design and code	**Projects (imitate, innovate, invent, remix)** There are different ways to scaffold learning in projects. This process can be applied to programming projects; • Using example projects e.g. the Guided 2Code activities. • Completing the challenges at the end of each guided activity. • Free code✓ • Create a project that imitates a high-quality exemplar. • Remixing ideas. • Independently creating a brand-new program.
	Tinkering	Use Free code Gorilla to access the full suite of 2Code objects and commands ✓ Use Free code to play and explore freely.

In Literacy, some teachers follow a progression that scaffolds learning to write texts. At first pupils read lots of examples of the genre of text they are going to create. Then they create an ***imitation*** of an example text. Next, they create a variation of the text (***remix and innovate***). Finally, they get to ***inventing*** a brand-new version.

Note: To force links within this document to open in a new tab, right-click on the link then select 'Open link in new tab'.

Year 4 – Medium Term Plan

Lesson	Aims	Success Criteria
Lesson 1 – Review the design, code, test, debug process	• To review coding vocabulary. • To use a sketch or storyboard to represent a program design and algorithm. • To use the design to create a program.	• Children can use sketching to design a program and reflect upon their design. • Children can create code that conforms to their design.
Lesson 2 – If/Else statements	• To introduce the If/else statement and use it in a program. • To create a variable. • To explore a flowchart design for a program with an if/else statement • To create a program which responds to the If/else command, using the value of the variable.	• Children can create an 'If/else' statement. • Children understand what a variable is in programming. • Children can set/change the variable values appropriately. • Children can interpret a flowchart that depicts an if/else flowchart.
Lesson 3 – Repeat Until	• To create a program with a character that repeats actions. • To use the Repeat Until command to make characters repeat actions. • To program a character to respond to user keyboard input.	• Children can show how a character repeats an action and explain how they caused it to do so. • Children can make a character respond to user keyboard input.
Lesson 4 – Making a timer	• To make timers and counting machines using variables to print a new number to the screen every second.	• Children can explain what a variable is when used in programming. • Children can create a timer that prints a new number to the screen every second. • Children can explain how they made their program change the number every second.
Lesson 5 - Making a control simulation	• To explore how 2Code can be used to investigate control by creating a simulation.	• Children can create an algorithm modelling the sequence of a simple event. • Children can manipulate graphics in the design view to achieve the desired look for the program. • Children can use an algorithm when making a simulation of an event on the computer.
Lesson 6 – Decomposition and Abstraction	• To know what decomposition and abstraction are in computer science. • To take a real-life situation, decompose it and think about the level of abstraction. • To design a decomposed feature of a real-life situation.	• Children can make good attempts to break down their aims for a coding task into smaller achievable steps. • Children recognise the need to start coding at a basic level of abstraction to remove superfluous details from their program that do not contribute to the aim of the task.

Lesson 1 – Review the design, code, test, debug process

Aims

- To review coding vocabulary.
- To use a sketch or storyboard to represent a program design and algorithm.
- To use the design to create a program.

Success criteria

- Children can use sketching to design a program and reflect upon their design.
- Children can create code that conforms to their design.

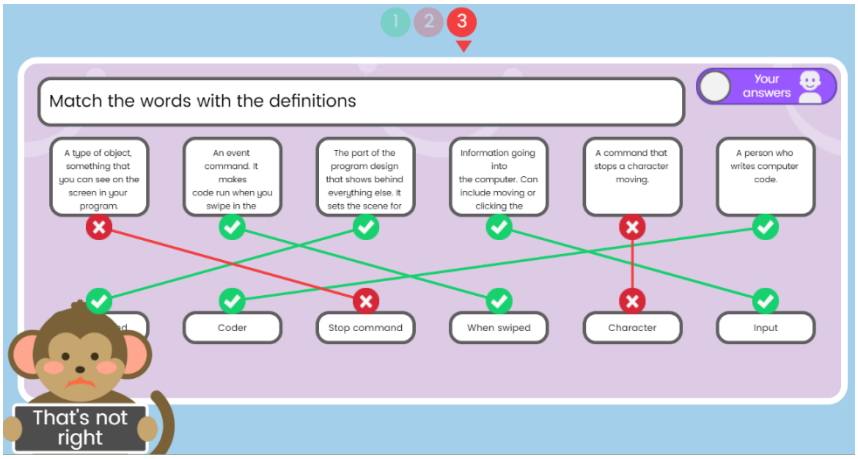
Resources

Unless otherwise stated, all resources can be found on the main unit 4.1 page. From here, click on the icon to set a resource as a 2do for your class. Use the links below to preview the resources; right-click on the link and 'open in new tab' so you don't lose this page.

- Coding Vocabulary Quiz 3
- Program Design examples to be displayed on the board.
- Plain paper for sketching a design or printed storyboard templates.
- Save a 2Write file called 'Coding Hints needed' in a shared class folder.
- (Optional) Vocabulary flash cards. The Teacher flash cards have been created in such a way that you can print them on A4 paper, cut them to size, fold them in half and glue them together.
- (Optional) Exercise books to be used as 2Code workbooks for recording coding exercises, if desired.

Activities

1. Use the quiz as a class. It is set up so that you attempt all questions and then click the Hand in button to check the answers. Click 'OK' to see which are correct and incorrect:



2. You can use the vocabulary cards to find the answers and display in the classroom.
3. Put Free Code Gibbon on the board. Review how to add objects in 2Code by going in to Design Mode. Drag a car and an animal into the Design View.
4. Ask children to sketch a simple design for a program in which two objects perform actions. They should detail the types of objects, the actions that the objects will perform and the events which will

cause these actions to occur. Use the design examples document to remind children of the types of designs they have created in the past. They have made programs in the past that can do the following:

- Characters that respond to being clicked.
- Characters that respond to collisions.
- Using timers to repeat actions.
- Using If statements.

5. Children may want to look over their programs from the past to refresh their memory on some aspects if they have not used 2Code for a while. The Gibbon activities might provide further reminders.
6. Once they have designed. They should rate their confidence at being able to code their design – they could highlight this is red/amber/green on their design documents.
7. Children should then open the 2Write file and add the things that they are less confident at being able to code to this. For example, 'making my character only if the other character hides'. This will provide you with information about specific coding structures to review over the next few lessons where possible.
8. Children should begin to turn their design into code and save and test it regularly.

Lesson 2 – If/Else statements

Aims

- To introduce the If/else statement and use it in a program.
- To create a variable.
- To explore a flowchart design for a program with an if/else statement
- To create a program which responds to the If/else command, using the value of the variable.

Success criteria

- Children can create an 'If/else' statement.
- Children understand what a variable is in programming.
- Children can set/change the variable values appropriately.
- Children can interpret a flowchart that depicts an if/else flowchart.

Resources

Unless otherwise stated, all resources can be found on the main unit 4.1 page. From here, click on the icon to set a resource as a 2do for your class. Use the links below to preview the resources; right-click on the link and 'open in new tab' so you don't lose this page.

- Vocabulary flash cards.
- Gibbon Night and Day guided activity.
- Night and Day Flowchart set this as a 2do for the class.

Activities

1. Ask children what **selection** means in coding. You can remind them that the **if statement** is a **command** that they have used to introduce **selection** in their programs. Review If statements. The children first saw these when doing the Gibbon Guard the Castle activity (lesson 3 in year 3 unit 3.1). Look at the flash card definition; can children remember how If statements can be used?

   Selection
   This is a conditional/decision command. When selection is used, a program will choose a different outcome depending on a condition, for example; "repeat"; "repeat until"; "if/else".
   If
   A conditional command. This tests a statement. If the condition is true, then the commands inside the block will be run.

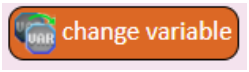
2. Review the term **Variable**; the children used it to create timers in lesson 4 last year (unit 3.1). They might remember doing the variable role play in that lesson.

   Variables are like boxes in which the computer can store information. To find the information in the box, each box should be labelled. Therefore, each variable (each of our boxes) needs to have a **name**. The **name** should be **something that helps you remember what it is**. The information inside the box is called the **Variable Value**. The user, the program or another variable can change this Variable Value.

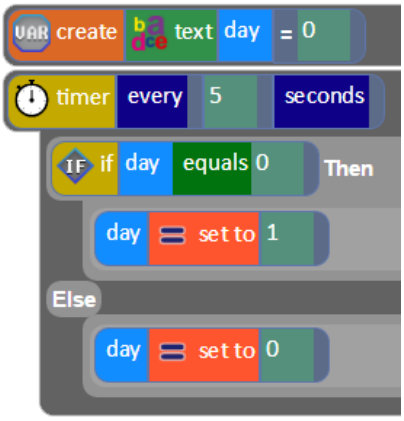
3. Open the program Night and Day from the Gibbon activities. Watch the video for step 1 and do step 1 as a class. The variable 'day' will be set to 0 when it is night-time, and 1 when it is daytime.
4. Move on to Step 2 where you must add a timer and an If/else statement. First remind the class how to add a timer that is called **every** 5 seconds.
5. Next, introduce the If/else statement:

> **If/Else**
> A conditional command. This tests a statement. If the condition is true, then the commands inside the 'if block' will be run. If the condition is not met, then the commands inside the 'else block' are run.

6. Drag the 

inside the timer and discuss how to select options for the code so that different code runs depending upon the value of the variable 'day'. Also discuss how to change the value of the variable by dragging the 

block into the if/else statement.



7. Read the resulting code, it says:
   - 'If the variable "day" is set to 0 then set it to 1, otherwise (else) set it to 0.'
8. The if/else statement is inside the timer which means that every five seconds the variable's value will change.

9. In the next step, the program should respond to the variable by changing the colour of the sky and making the sun show or hide.

10. Once this step is completed, the children are asked to debug the monkey's code. Can they do this together as a class?

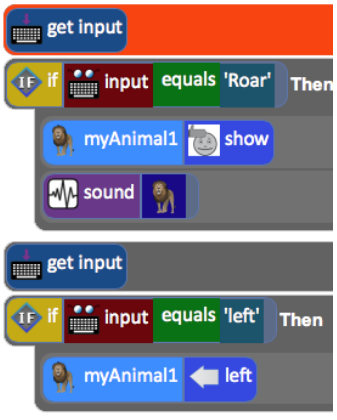
11. Now, open the flowchart example on the board. Do children know what it is. They might have used flowcharts in year 3. Trace the flowchart together, what coding structure does it represent? **An if/else statement.**

12. Show children how to edit the chart, they should try to avoid restructuring the whole chart as this can be quite fiddly and they won't have time to code:

- Double-click on a box to edit the text.
- Drag a square over multiple boxes and lines to select them and then move them if necessary.
- Remove parts by clicking on them and then clicking 'delete' on the keyboard.
- Draw the arrows by clicking on a box, then clicking on the pale blue arrow and dragging to the box to join.

13. Children should open this flowchart from their 2dos and plan a program with a character that repeats an action until they input a command to stop it? For example, using a timer to repeat until a character is clicked on. They should then code, save and test their program.

    If they finish this, they can also try creating a program with an animal that moves and makes sounds when using text commands. For example, asking the user to enter an animal name from a selection, and then making an appropriate sound and/or an image appear, based on the user's answer (using If/else functions).



Lesson 3 – Repeat Until

Aims

- To create a program with a character that repeats actions.
- To use the Repeat Until command to make characters repeat actions.
- To program a character to respond to user keyboard input.

Success criteria

- Children can show how a character repeats an action and explain how they caused it to do so.
- Children can make a character respond to user keyboard input.

Resources

Unless otherwise stated, all resources can be found on the main unit 4.1 page. From here, click on the icon to set a resource as a 2do for your class. Use the links below to preview the resources; right-click on the link and 'open in new tab' so you don't lose this page.

- Vocabulary flash cards.
- Repeat and sequence example program.
- Blank printable storyboards for designing.

Activities

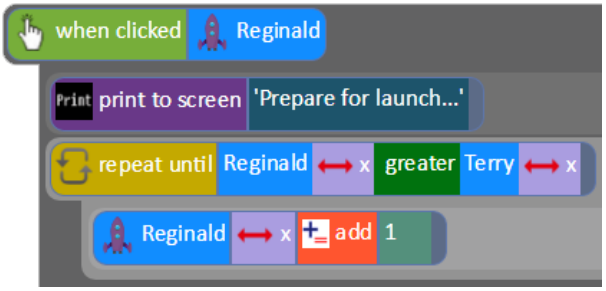
1. In this lesson we will be reviewing and learning some vocabulary relating to programming. On the board, go through the words Sequence, Selection, Repeat, Repeat Until, Input and Output. Children could play memory games with the child flash cards to learn the vocabulary. Children can also review the vocabulary using the quizzes found in the quizzes section on the bottom of the 2Code main screen.

2. Open the example program Repeat and Sequence example activity. First, read the code and try to work out what the program will do. Children could jot down their ideas.

3. Then **execute** (run) the program. You need to click on the rocket and it begins blasting off, then a flock of sheep appears, and the user must type in 'Hide' or 'Panic'.

4. Look at Code View and point out the 

command. Can they 'read' the code to see what this command is doing? When the user clicks on Reginald (the rocket), a message is printed to the screen – Prepare for Launch – then Reginald will move right (adding 1 to the X property) this repeats until the X is greater than the X position of Terry.



5. Ask children what/who is Terry? And what is the purpose of this piece of code? Terry is the launch pad (you can work this out by looking for the object called Terry in Code View). The piece of code moves the rocket onto the Launchpad.

6. Now look at the next part of the code.

7. Identify the following parts in the code; after three seconds (timer), there is a sound, a message is printed to the screen and the rocket goes up.

8. Identify the following parts in the code; after another second (another timer), the rocket stops, and three sheep appear (show). Note that, in the design, when you click on the sheep and look at the properties box on the left-hand side, they are initially set to 'Hide'.

9. Identify the following parts in the code; the user is asked whether to Hide or Panic. The If/else statement either hides the sheep or makes them run around. How does it do this? If the user types in something other than Hide or Show, what should happen? Try it.

10. Children should plan a program, using the storyboards or 2Chart, with the following task specification:

    Task: Create a short program that uses Repeat Until and If/else commands.

11. Once they have created written plans, they should add annotation showing the required objects and actions/commands. They should also annotate their confidence in being able to turn the aspects of their design into code and seek advice or refine their plans if they are not confident about coding the design. They will need to decide whether the design is too ambitious for their current skill level or if there are things that they have learnt but are not confident about and need a bit of help.

12. They can then code, save, test and debug.

Lesson 4 – Making a timer

Aim

- To make timers and counting machines using variables to print a new number to the screen every second.

Success criteria

- Children can explain what a variable is when used in programming.
- Children can create a timer that prints a new number to the screen every second.
- Children can explain how they made their program change the number every second.

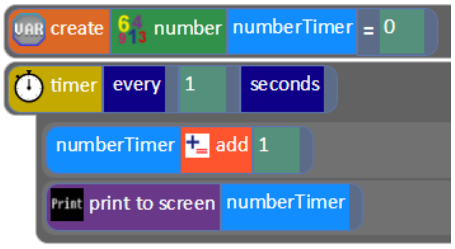
Resources

Unless otherwise stated, all resources can be found on the main unit 4.1 page. From here, click on the icon to set a resource as a 2do for your class. Use the links below to preview the resources; right-click on the link and 'open in new tab' so you don't lose this page.

- Example program Variable example code to be opened on the whiteboard to demonstrate..

Activities

1. Children used variables to create a timer in Year 3 (lesson 5 of unit 3.1). Can they remember how to do this?
2. Why is it important for the variable to have a sensible name? (It makes it easier to keep track of as the program becomes more complex.)
3. Open the example program and ask children to read the code and predict what the program will do. **Execute** (run/play) the code to see if their predictions were correct.



4. Direct the children to watch the bottom left-hand corner of the screen as the timer counts. This is the **Variable Watch**. It tells us what variables exist in the program and what they are doing. Variable Watches are common in programming environments to help programmers understand what the computer is doing and to debug their code.
5. Children should now to make their own timers. Once made, the children can experiment with number variables and see what they can make their programs do using them, for example they could try to make counting machines that count times tables.
6. They could also add a visual effect tot heir program, perhaps an object whose scale increases as time goes no until it reaches a particular size. Then something else happens.
7. When they have had enough time experimenting with these, they can try the counting machines in Coding Principles and practise with number variables (on the 2Code main page in the section called Coding Principles).

Lesson 5 - Making a control simulation

Aim

- To explore how 2Code can be used to investigate control by creating a simulation.

Success criteria

- Children can create an algorithm modelling the sequence of a simple event.
- Children can manipulate graphics in the design view to achieve the desired look for the program.
- Children can use an algorithm when making a simulation of an event on the computer.

Resources

Unless otherwise stated, all resources can be found on the main unit 4.1 page. From here, click on the icon to set a resource as a 2do for your class. Use the links below to preview the resources; right-click on the link and 'open in new tab' so you don't lose this page.

- Video of the UK traffic light sequence to view as a class.
- Traffic Light Algorithm vocabulary – print this so that children can see a copy while they watch the video.
- (optional) Set the writing file Traffic Lights Algorithm as a 2do. Some children might find this more accessible than using 2Chart to create a flowchart.

Activities

1. Remind children of the word **algorithm** and its definition;

   An ***algorithm*** is a precise step by step set of instructions used to solve a problem or achieve an objective.

2. Ask children to give some examples of everyday processes that you could write an algorithm for. For example, making breakfast, getting dressed, the journey to school.

3. Explain that they are going to be creating an algorithm for a traffic light sequence and then using the algorithm to write a computer program that ***simulates*** this sequence. Discuss the meaning of the words **simulates/simulation.**

   A **simulation** is a model that represents a real or imaginary situation. Simulations can be used to explore options and to test predictions.

4. Display the traffic light vocabulary somewhere that children can see, suggest that some of these words might help them when writing the algorithm.

5. Ask the children to take notes whilst watching the traffic light video that will enable them to write the algorithm. You may need to play the video several times.

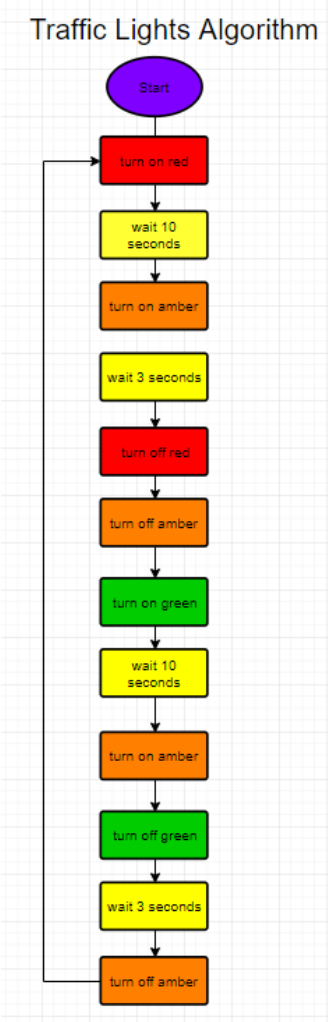
6. Show children how to open 2Chart from the Tools area of Purple Mash. Recommend that they start with a blank sheet or edit the simple sequence example (they will need to make the boxes smaller). **NB** You might want to direct some children to use the writing template linked in the resources section – you can compare the resulting algorithm using the two methods.

   - Show children how to change the text and add a title by double-clicking on the existing text.

- Show them how to remove unwanted parts by clicking on them and pressing the delete key.
- How to add arrows by clicking on a box, clicking on the pale blue arrow that appears and dragging it to the box to be joined. The arrow lines can then be edited by dragging the blue dots if necessary.
- It can be fiddly; the children do not have to get it looking perfect, the information it shows is the main thing.
- Remind them to save regularly.

7. Give children 10 minutes to try to make a flowchart or diagram to show the algorithm. It should look something like this:

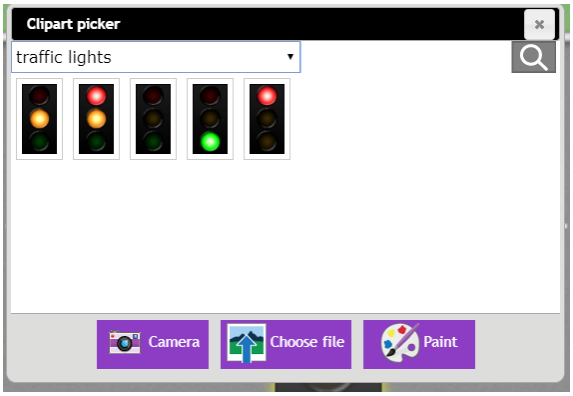
Turn on red
Wait 10 seconds
Turn on amber
Wait 3 seconds
Turn off red
Turn off amber
Turn on green
Wait 10 seconds
Turn on amber
Turn off green
Wait 3 seconds
Turn off amber
Repeat all forever



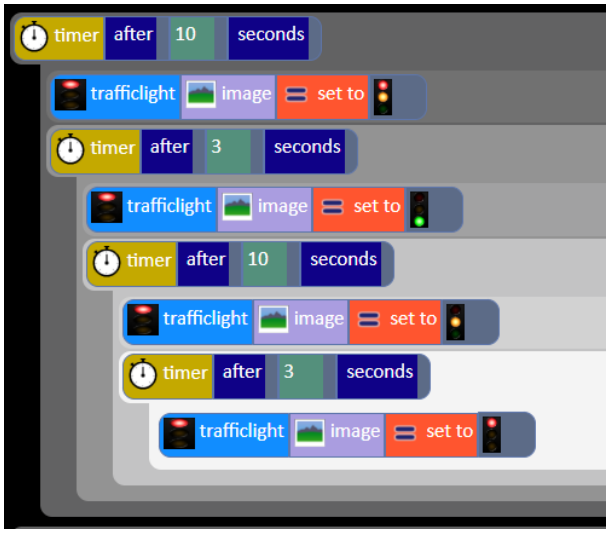
Note: At a pedestrian crossing the sequence is different. For the moment, we are concentrating on traffic lights that are not at a pedestrian crossing.

8. Open Free Code Gibbon. In design view, change the background to one of the ready-made road layouts (any will do).

9. Drag in any object, rename it trafficlight.
10. In the clipart picker is a section called 'traffic lights'. Set the image to the correct one for the start of the sequence. They will be making the program by coding their object to change image at the correct intervals.
11. Save the file and switch to code mode.



12. Now it's time to code.
13. The code will need to repeat forever, what command is needed to do this? Children might suggest the repeat or repeat until commands but the rest of the algorithm uses timers. Do they remember what they learnt about using timers and repeat together in the past (see year 2, lesson 2 and year 3, lesson 5). In 2Code timers do not work properly when used with repeat commands as the repeat tries to run the code as quickly as possible while the timer tries to slow things down so you get unexpected results. This means that children need to use timers to make the sequence repeat.
14. The way to code such a sequence in 2Code is to code the first sequence then write code that will repeat this sequence. Give children a bit of time to try coding the first sequence, assuming the initial traffic light image is the red light on. They should save, test and debug. Then bring the class back together to see what they have achieved.
15. They will need to ensure that the timers are all **nested** (inside each other). Here is an example using the algorithm above:

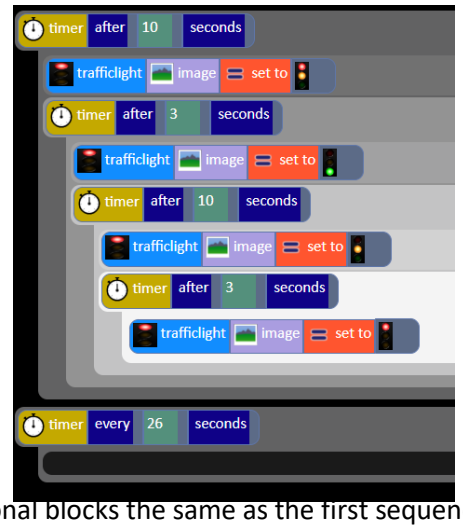


16. Now we want to make the sequence repeat. You can do this using a timer that repeats every x seconds. Can children work out how many seconds x should be? To do this add up all the time intervals 10+10+3+3= 26.

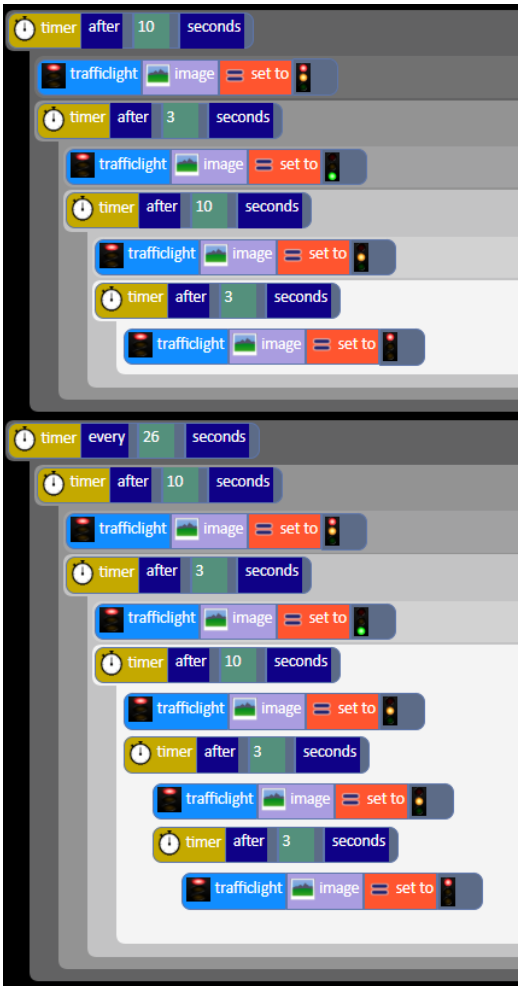
17. If you just add this line of code and drag in the existing code, then the sequence will only start after the first 26 seconds is over. Instead, leave the code as it is and drag in a new timer below the existing code set it as follows:

Note the code in the timer says

every rather than ***after***.



18. Now copy the sequence into this timer by adding additional blocks the same as the first sequence:



19. Now give the children time to write this code and test it.

20. An advantage of the development of simulations is to allow real-life situations to be tested out before they are built in the 'real world'.

21. Ask the children to imagine the scenario that this light was used at a cross-road junction and it was found that the traffic was building up too much at the red light.

22. Can they adapt the code so that the green light is on for longer and the red light for a shorter time?
23. They might need to re-write the algorithm to help them work out the timing.

Extension

24. The sequence on a pedestrian crossing is different because it is started when a pedestrian presses a button.
25. Children could try to simulate this type of crossing as well.
26. There is a button object in 2Code.
27. A character object could be used for the red and green man; they allow their images to be changed in the code and uploaded from a computer. Children could find red and green man pictures online to use.

Lesson 6 – Decomposition and Abstraction

Aim

- To know what decomposition and abstraction are in computer science.
- To take a real-life situation, decompose it and think about the level of abstraction.
- To design a decomposed feature of a real-life situation.

Success criteria

- Children can make good attempts to break down their aims for a coding task into smaller achievable steps.
- Children recognise the need to start coding at a basic level of abstraction to remove superfluous details from their program that do not contribute to the aim of the task.

Resources

Unless otherwise stated, all resources can be found on the main unit 4.1 page. From here, click on the icon to set a resource as a 2do for your class. Use the links below to preview the resources; right-click on the link and 'open in new tab' so you don't lose this page.

- Some examples of simple board games such as snakes and ladders, chess and solitaire.
- Decomposition and Abstraction writing frame -set this as a 2 do or print copies

Activities

1. In coding lessons, children have been using decomposition and abstraction to achieve the aims of the task. Do they have any idea what these words mean? (Probably not).

2. Explain that **decomposition** is breaking a task into its component parts so that each part can be coded separately and brought together in the program. When they coded Day and Night, they broke this down into what happens in day and what happens in night. In the timer tasks, they broke this down into making a basic timer and then adding things that happened dependent upon the timer. If you were designing a program that simulated a board game, you could split it into throwing the dice, moving the piece, winning, losing, activity in between. When you create more complex programs and their algorithms it helps to consider parts at a time

3. **Abstraction** is removing unnecessary details to get the program functioning. In the traffic light task, the road layout was not relevant initially, the weather and time of day didn't matter, and we didn't add any cars to the program. We made a simulation at a basic level (a high level of abstraction).

4. Today, you are going to try these two processes to code part of a simple board game. Children could choose the game, or you could allocate different groups to different games.

5. Show the children the writing frame and ask for some examples relating it to any of the games that you have. The writing frame has a word bank that will also help with this.

6. Children should fill in the sheet framework for their game and then use this to code one part of the game e.g. throwing the dice, moving the piece, keeping tabs on which payer is winning. If children work in groups, each child or pair could try coding a different part. The aim isn't to finish coding the whole game but to experience how this process works in coding and how it leads towards the final program. If children are working in groups then they should decide upon the names for the objects in the game and the names of variables and try to be consistent between groups so in an ideal situation, all the parts could be combined into a program.

Assessment Guidance

The unit overview for year 4 contains details of national curricula mapped to the Purple Mash Units. The following information is an exemplar of what a child at an expected level would be able to demonstrate when completing this unit with additional exemplars to demonstrate how this would vary for a child with emerging or exceeding achievements.

Assessment Guidance	
Emerging	With support, children can turn a real-life situation into an algorithm for a program that has cause and effect (lesson 5) and use their algorithm to write simple programs using 2Code (lesson 1). Furthermore, they can identify errors within their programs and make logical attempts to fix it (all lessons). Pupils attempt to introduce repetition and selection into their code using timers and simple 'if statements' (lessons 2, 3 & 4). Children's use of these structures is experimental; they cannot always predict the outcome accurately or anticipate the structures required when planning their code. They have a developing idea that a variable can be used to store information in a program, in lesson 5 they can follow the examples but might struggle when applying this with their own ideas. Children can use the 'get input' command (lesson 2, step 14) to work with user input. Children's designs for their programs, show that they are thinking of the structure of a simple program in logical, achievable steps (lessons 1 & 2). Children can make good attempts to 'read' code and predict what will happen in a program which can help them to correct errors in their code. In lesson 5, children try to write the algorithm for the traffic lights but might miss some steps.
Expected	Children can turn a simple real-life situation into an algorithm for a program by deconstructing it into manageable parts (lesson 5 & lesson 6). Children's design shows that they are thinking of the required task and how to accomplish this in code using coding structures for selection and repetition (lessons 2 & 3). Children can identify an error within a program that prevents it following the desired algorithm and then fix it (lesson 4), they apply these techniques to their own code to fix bugs. Children's use of timers to achieve repetition effects are becoming more logical and are integrated into their program designs (lessons 2 & 4). They understand 'if statements' for selection and combine these with other coding structures including variables to achieve the effects that they design in their programs (lessons 2 & 4). Their design demonstrates their growing understanding of when a coded solution will require repetition e.g. in lesson 3, within 2Code example 'Repeat and Sequence' children can see that the position of the rocket is changed repeatedly until it is in line with the rocket launch pad. They can explain the new command 'Repeat Until'. They make use of user input (lesson 2, step 14) and outputs such as 'print to screen' (lesson 3) as well as sound and movement of objects. They understand how variables can be used to store information while a program is executing (lesson 4) and make attempts to use and manipulate the value of variables. Children's designs for their programs, show that they are thinking of the structure of a simple program in logical, achievable steps with attention to specific events that initiate specific actions (lesson 1, step 4 and lesson 3, step 7). Children can 'read' others' code and predict what will happen in a program which helps them to correct errors (lessons 2 & 3). They can also make good attempts to fix their own bugs as their coding becomes more complex (lesson

Assessment Guidance	
	5 & 6). In lesson 5, they make a good attempt to write the traffic light algorithm and can easily recognise any errors in their algorithm when discussing it as a class. Most children can create programs which accomplish a specific goal utilizing a variety of media such as images (including photos), sounds and animation effects. Children can manipulate graphics in the design view of 2Code to achieve the desired look for the program (Unit 4.1 Lessons 1,5 & 6). Children can interpret the flowcharts used to represent if/else (lesson 2) and create their own when planning their programs. In lesson 5, children create a flowchart of the sequence of traffic lights and use this to create a coded simulation of the traffic lights. They can demonstrate the need for 'repeat' and as a result can use timers to make a repeating sequence. Children can plan simple simulations using software such as 2Chart, adapting premade templates such as 'repeat until' as required.
Exceeding	Children are attempting to turn increasingly complex real-life situations into algorithms for a program by deconstructing the situation into manageable parts (lessons 5 and 6). Children's design shows that they are thinking of the required task and how to accomplish this in code using coding structures for selection and repetition and variables (lessons 2 & 3). Children can identify an error within a program that prevents it following the desired algorithm and then fix it (all lessons). Children make intuitive attempts to debug their own programs as they increase in complexity. Pupils realise the constraints of creating purely sequential programs and intuitively grasp the concepts of selection (lesson 2), repetition (lesson 3) and variables (lessons 2 and 4). Children like to challenge themselves to combine these with other coding structures to achieve the effects that they design in all their programs (all lessons). Their designs are ambitious but logical and achievable. Children's designs for their programs, show that they are absorbing new knowledge of coding structures such as 'if' statements, repetition and variables to think of their programs in logical, achievable steps (lesson 1, step 4 and lesson 3, step 7). Children can 'read' others' code and predict what will happen in a program which helps them to correct errors (lessons 2 & 3). They can also make good attempts to fix their own bugs as their coding becomes more complex (lessons 5 & 6). In lesson 5, they intuitively recognise the required steps of the traffic light algorithm and can then adapt this to complete the extension activity.