purple mash

Computing Scheme of Work

Unit 3.1 - Coding

Year Group: 3
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 and 2. 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 timers 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?

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	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.
Least scaffolded	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 3 – Medium Term Plan

Lesson	Aims	Success Criteria
1: Review Previous Coding	• To review coding vocabulary that relates to Object, Action, Output, Control and Event. • To use 2Chart to represent a sequential program design. • To use the design to write the code for the program	• Children can create a design that represents a sequential algorithm. • Children can use a flowchart design to create the code. • Children can explain what Object, Action, Output, Control and Event are in computer programming.
Lesson 2 – A Physical System	• To design and write a program that simulates a physical system.	• Children can explain how their program simulates a physical system, i.e. my vehicles move at different speeds and angles. • Children can describe what they did to make their vehicle change angle. • Children can show that their vehicles move at different speeds
Lesson 3 – If commands	• To look at the grid that underlies the design and relate this to X and Y properties. • To introduce selection in their programming by using the if command. • To combine a timer in a program with selection.	• Children can make use of the X and Y properties of objects in their coding. • Children can create an if statement in their program. • Children can use a timer and if statement to introduce selection in their program.
Lesson 4 - Variables	• To understand what a variable is in programming. • To use a variable to create a timer	• Children can explain what a variable is in programming. • Children can explain why variables need to be named. • Children can create a variable in a program. • Children can set/change the variable values appropriately to create a timer.
Lesson 5 - Repetition	• To create a program with an object that repeats actions indefinitely. • To use a timer to make characters repeat actions. • To explore the use of the repeat command and how this differs from the timer.	• Children can show how their character repeats an action and explain how they caused it to do so. • Children are beginning to understand how the use of the timer differs from the repeat command and can experiment with the different methods of repeating blocks of code. • Children can explain how they made objects repeat actions.
Lesson 6 - Debugging	• To know what debugging means. • To understand the need to test and debug a program repeatedly. • To debug simple programs. • To understand the importance of saving periodically as part of the code development process.	• Children can explain what debug (debugging) means. • Children have a clear idea of how to use a design document to start debugging a program. • Children can debug simple programs. • Children can explain why it is important to save their work after each functioning iteration of the program they are making.

Lesson 1 – Review Previous Coding

Aims

- To review coding vocabulary that relates to Object, Action, Output, Control and Event.
- To use 2Chart to represent a sequential program design.
- To use the design to write the code for the program

Success criteria

- Children can create a design that represents a sequential algorithm.
- Children can use a flowchart design to create the code.
- Children can explain what Object, Action, Output, Control and Event are in computer programming.

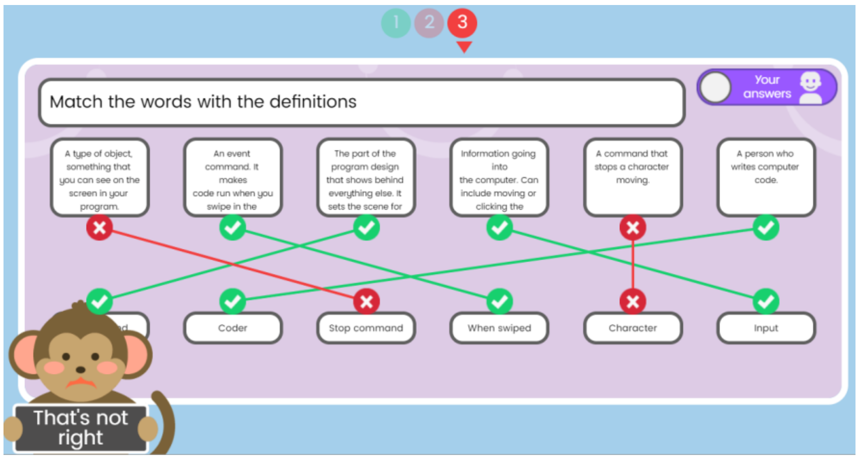
Resources

Unless otherwise stated, all resources can be found on the main unit 3.1 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 2
- Flowchart - example algorithm. You might wish to print a copy of this to refer to while coding.
- 2Code Freecode Gibbon (this is found on the main 2Code page).
- (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:



Match the words with the definitions

A type of object, something that you can see on the screen in your program.	An event command. It makes code run when you swipe in the	The part of the program design that shows behind everything else. It sets the scene for	Information going into the computer. Can include moving or clicking the	A command that stops a character moving.	A person who writes computer code.
	Coder	Stop command	When swiped	Character	Input

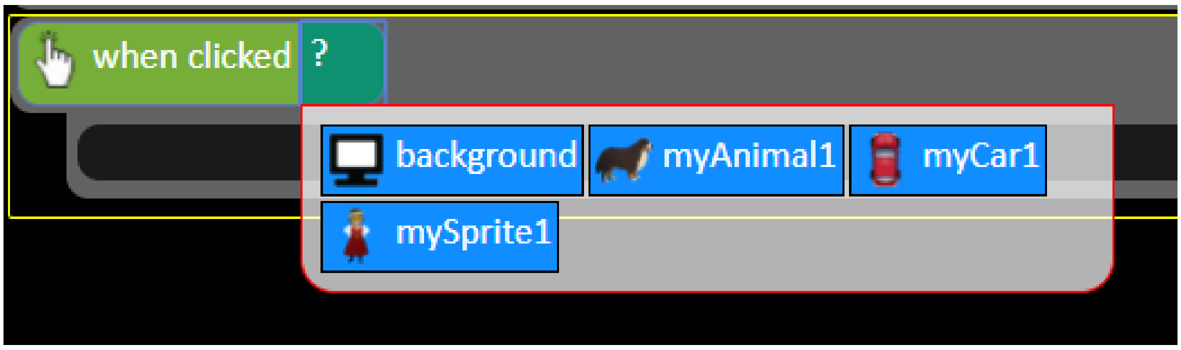
You can use the vocabulary cards to find the answers and display in the classroom.

2. Review the word algorithm – how did the children show their algorithms in Y1 and 2? They might have used design diagrams, storyboards or lists.

3. Show the children the flowchart example made using 2Chart. Explain that this is another way to show the algorithm for a program.

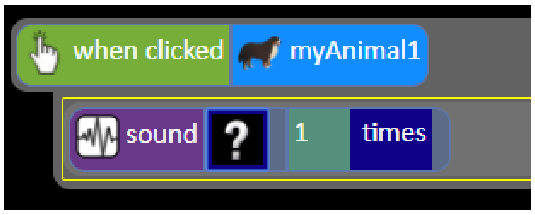
4. Read the flowchart together. Can the children see that there are two objects; an animal and a character. What should the objects do?
    - The animal should make a sound when clicked on.
    - When the background is clicked, the animal should move right, when the animal collides with the character, the animal should stop, and the character should say 'ouch!'.
5. What is the difference between the different coloured boxes?
    - The purple ovals show the start of the sequence; often the initial **input.**
    - The green boxes are **actions** that happen in a sequence.
    - The orange diamond is a **decision**, if the answer is yes, then the yes arrow is followed.
6. Go to the main 2Code Page and show children where to find the Free Code Gibbon Icon by scrolling down. Children could do this on their own screens at the same time or you could demonstrate.

7. Open Free Code Gibbon on the board. In years 1 and 2, children used Chimp level so the options on the screen might look a bit different to the children. Review how to add objects in 2Code by going into (Design Mode). Drag a car, a character and an animal onto the background.
8. Return to Code View by clicking .
9. Remind children how to save their work and briefly discuss the need to save regularly so they always have a saved, working version of their program to go back to.
10. Now we are going to try and write code that follows the algorithm.
11. Drag in a When Clicked code block and review how this event works:

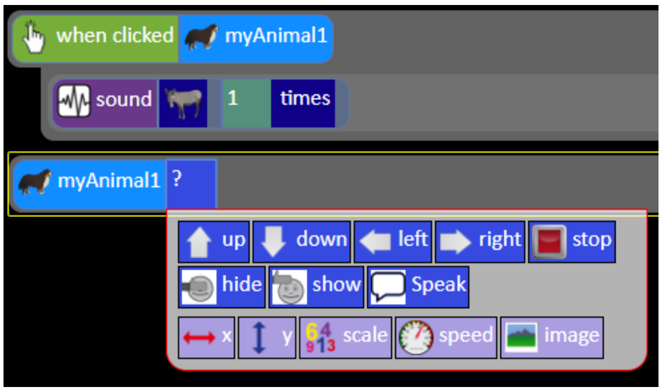


12. According to the **algorithm**, the **object** that we want to trigger the **action** is clicking on the animal. Then we need to code what happens when we click on the object. This is called **output**. What should the **output** be?

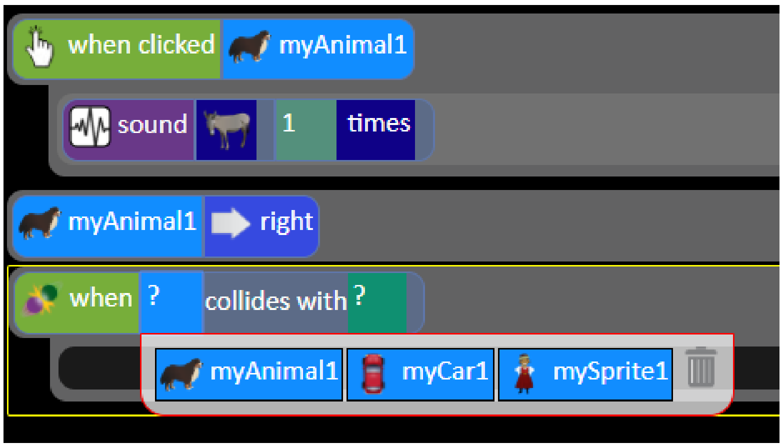
13. We are going to make the object make a sound when clicked so drag a Sound block into the grey box under When Clicked so that it is indented slightly. Remind children that this indentation makes the output happen due to the input (clicking the animal).



14. Click on the question mark and select a sound.
15. Save and then test the program so far. Does the animal make a sound when you click on it?
16. What else does the algorithm say should happen and how can we make this happen?
17. The animal should move right so drag the animal object into the code window below the existing code.

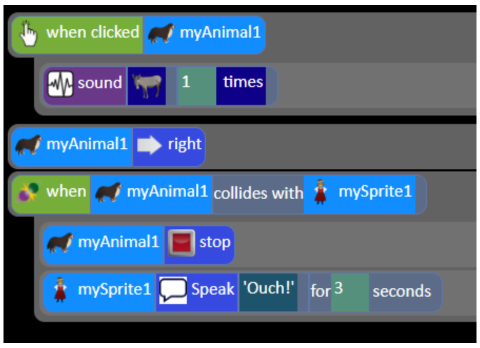


18. A menu will pop up; ask the children what it is called (the **action** menu).
19. Select the action 'right'. The save and test the code again.
20. What do we need to code next? There should be some code when the animal collides with the character.
21. Drag in a collision detection block:



22. Ask the children which options to select so that the line or code will do something when the animal collides with the character.

23. What should happen next according to the algorithm? When this happens, the character should say something, and the animal should stop. Ask the children to explain how to write the code for this.



```
when clicked myAnimal1
    sound 1 times
myAnimal1 right
when myAnimal1 collides with mySprite1
    myAnimal1 stop
    mySprite1 Speak 'Ouch!' for 3 seconds
```

24. Save, test and debug (if necessary) the program.
25. What should the car do? Nothing as no action is detailed in the algorithm.
26. Children should now be given some time to explore free code Gibbon, reminding themselves of things that they did in years 1 and 2 and trying out new commands that they notice.
27. One thing that they could explore is the different possible actions of the different object types. They could compare the possible actions of the vehicle and the character, for example.
28. If the children have workbooks, they can print their code and write about why they chose the commands they did and why they put them in that specific order.

Lesson 2 – A Physical System

Aims

- To design and write a program that simulates a physical system.

Success criteria

- Children can explain how their program simulates a physical system, i.e. my vehicles move at different speeds and angles.
- Children can describe what they did to make their vehicle change angle.
- Children can show that their vehicles move at different speeds

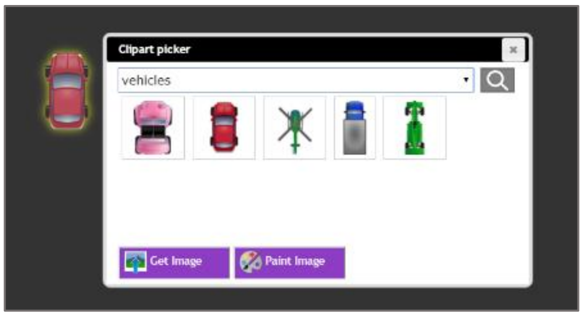
Resources

Unless otherwise stated, all resources can be found on the main unit 3.1 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.

- Properties video
- 2Code Freecode Gibbon (this is found on the main 2Code page).
- (Optional) print storyboard templates for program design.
- (Optional) Exercise books to be used as 2Code workbooks for recording coding exercises, if desired.
- (Optional) Self-Assessment templates:
  - Pdf for printing
  - Writing template

Activities

1. Today we will be creating a new program to do something specific that simulates a physical system. Ask the children if they know what this means. Take some suggestions.
2. It means creating a program where the **objects** behave as they would in the real world. We will be using vehicle objects and they should move at different speeds and should change angles.
3. Demonstrate opening free code Gibbon and go into Design Mode and drag in a vehicle object.

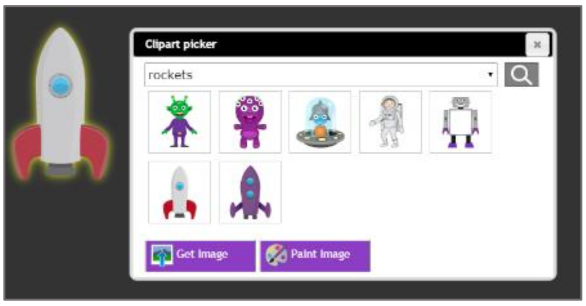


4. Double-click on the vehicle and show children that they can change it into something else if they like. They can even draw their own image or upload one from their computer, but it will retain the properties of a vehicle. This is important for them to be able to change the speed and angle. The Properties of Objects video explains this.

5. Exit Design Mode and drag the vehicle into the black code box. Show the pop-up menu and ask children where the angle and speed options are.

6. For the program that they are going to make, children should include at least three vehicles that travel at different speeds, and at least one vehicle should change angle. The Chimp activity, 'Vehicles', shows cars moving at different speeds.

7. Children should work in pairs to discuss what program they want to design and should create a design



document before coding. Children can use their 2Code workbooks to write down notes that will help them plan their programs or they could create a labelled diagram or storyboard. They should decide:

- Which physical system they are simulating?
- How many vehicles they will include in their program.
- The steps of their algorithm - What their program should do?

8. Once children have their programs planned out, they can try to create it in Free Code Gibbon.

9. For extension, children can look at the Gibbon activity 'Vehicles 2', which adds other elements to the movement of the vehicles.

10. It is also useful for children to spend some time evaluating how successful they were in creating their programs in Free Code Gibbon. Did planning help? Did their tinkering in the previous lesson using Free Code give them ideas? Children can use their workbooks or the self-evaluation files to record this.

Lesson 3 – If commands

Aims

- To look at the grid that underlies the design and relate this to X and Y properties.
- To introduce selection in their programming by using the if command.
- To combine a timer in a program with selection.

Success criteria

- Children can make use of the X and Y properties of objects in their coding.
- Children can create an if statement in their program.
- Children can use a timer and if statement to introduce selection in their program.

Resources

Unless otherwise stated, all resources can be found on the main unit 3.1 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.

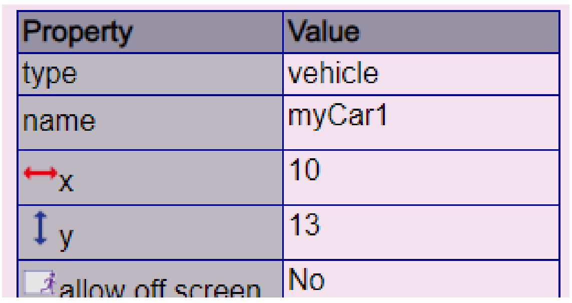
- (Optional) Vocabulary flash cards.
- Knights Castle flowchart – set this as a 2do for the class.
- Guided coding activity Guard the Castle (Gibbon) (this is found on the main 2Code page).
- Have printed storyboard templates available for program design.

Activities

1. Introduce the new vocabulary that they will be learning about today: 'if' statements.

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. Open free code Gibbon. Go into Design View and click the ▦ button in the bottom left. This makes a grid visible. Drag in a vehicle and look at the property window for it. You will see it has an X and Y position with a little icon showing which is which. Drag the vehicle to a different position and you will see that the properties change.



Property	Value
type	vehicle
name	myCar1
↔ x	10
↕ y	13
allow off screen	No

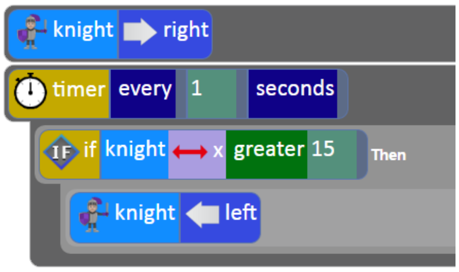
3. Work out where 0,0 is and the maximum X and Y by dragging the vehicle around. Give children X and Y positions and see whether they can make a good estimate as to where the vehicle should go. You could relate this to the context of coordinates and graphs.

4. Click on the button and change the grid size to a different size. See how this affects the X and Y positions. If you change the grid size after you have set up the screen design, it can mess things up so do this before you start coding.

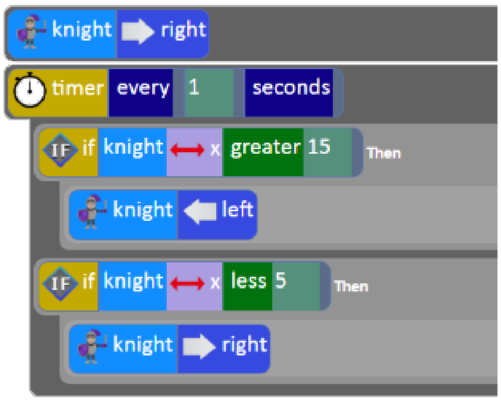
5. Briefly review how to make a character respond to a user's input on the keyboard.

6. Open the guided lesson 'Guard the Castle' in the Gibbon activities and do Step 1 together.

7. In Step 2, you must create a timer which checks the X position of the knight every second; if the knight's position is greater than 15, he should change direction. Can children explain how measuring the X position can decide whether the knight should change direction? Complete this step together explaining how to correctly insert the required code blocks and select options.


```
knight right
timer every 1 seconds
    if knight x greater 15 Then
        knight left
```

8. Do Step 3 together, adding another 'if' statement. Talk about how this should be indented at the same level as the first 'if' statement, so it is also inside the timer.


```
knight right
timer every 1 seconds
    if knight x greater 15 Then
        knight left
    if knight x less 5 Then
        knight right
```

9. Open the flowchart example for this activity in a different tab on and read it together to ensure that they understand it.

10. Do the debugging step of the activity and then watch the video for the challenge activity.

11. Children should now plan their challenge, either by adding to the flowchart or using a printed storyboard.

    - They should use at least one 'if' statement and timer in their program.
    - They should try to use the X and Y positions as part of their code.

Once planned, they should create, save, test and debug their code.

Lesson 4 - Variables

Aims

- To understand what a variable is in programming.
- To use a variable to create a timer.

Success criteria

- Children can explain what a variable is in programming.
- Children can explain why variables need to be named.
- Children can create a variable in a program.
- Children can set/change the variable values appropriately to create a timer.

Resources

Unless otherwise stated, all resources can be found on the main unit 3.1 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.

- 2Code Freecode Gibbon (this is found on the main 2Code page).
- 4 Mini/Individual whiteboards and whiteboard pens.
- (Optional) Vocabulary flash cards.

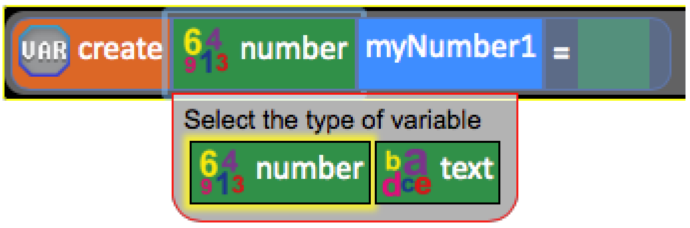
Activities

1. Today we will be working with variables. Here is the definition. You could display the flash card to remind children.

   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.

2. The following will help to explain the idea to the children. Choose two children to stand at the front of the class (they must have different names), give each child a board and a pen. Explain as follows:
3. Each child is a variable. They both have names, the first variable is called <insert child's name here – (for this example we'll use 'A')>, the other variable is called <insert other child's name here (for this example we'll use 'B')>.
4. I am going to store the value '2' in variable A(substitute child's name).
5. I am going to store the value '5' in variable B.
6. Ask both children to write their value on the whiteboard.
7. Now I am going to increase the value of variable A by 3 – Ask child A to rub out their value and write the new one.
8. Now I am going to multiply the value of variable B by 3 - Ask child B to rub out their value and write the new one.

9. Ask a new child to come and be another variable, introduce them to the class ‘this is variable <insert name (we’ll use C)>’.
10. I am going to set the value of variable C to the value of B times the value of A. Ask C to write their value.
11. Now I am going to set the value of A to 5. Get A and C to write their new values.
12. Now I am going to set the value of B to 2. Get B and C to write their new values.
13. Now I am adding a timer so that the value of A is going to increase by 1 every 5 seconds when I say go. See whether the variables can keep up with changing their values as you count off the seconds. Explain that the advantage for a computer is that it can do these calculations almost instantly.
14. Ask another child to come up to be a fourth variable called <insert child’s name (D)>.
15. Set the value of D to ‘The’. D should write their value. Explain that the previous variables were number variables but this one is a text variable.
16. Now set the value of D to ‘the current value ‘+ <space>+a random noun – get someone from the class to give you any noun. D should rewrite their value e.g. ‘The cat’ (talk about the need for the <space> otherwise it would be Thecat.
17. Now set D to ‘the current value ‘+ ‘<space> sat on the ’. D should rewrite their value e.g. ‘The cat sat on the’
18. Now set the value of D to ‘the current value ‘+ <space>+a random noun+’.’ – get someone from the class to give you any noun. D should rewrite their value e.g. ‘The cat sat on the fridge.’
19. Now say you are going to set the value of variable D to the current value multiplied by C. What should D write now? They should write ERROR as you can’t multiply a text variable by a number variable. (The + used with the text variable is really ‘and’).
20. So, variables need a name – usually they will have a name that explains what they are for to help you understand and debug your code, and a value that can change as the program runs. Variable can use the values of other variables.
21. Explain that you are closing the program now, so the variables should wipe off their values and return to their seats. Ask what the variable values will be the next time the program is opened; will they keep their current values? No, the values will reset to whatever the program initially sets them to.
22. Now open Free Code Gibbon on the whiteboard and look at the orange variable buttons in the menu on the left-hand side.
23. Drag Create Variable into the black code box. The drop-down menu only gives two options. Why is this? Remind children that, in 2Code, only options for variables are numbers or text.

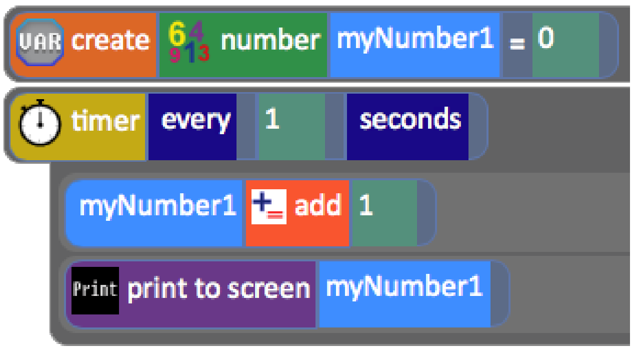


24. Choose Number. Give the variable a name – 'myNumber1'is fine for now. Ask children why we are naming the variable? What should the variable equal? We can either define the variable as a specific number or set it to Random. For this example, we will set the variable to 0.

25. Drag in a timer and change 'After' to 'Every'.

26. Drag in

and place it in the yellow box under Timer. We are using the timer as this is an easy way for us to get the variable to continue changing all the time. Select your variable from the drop-down menu. Set it to Add 1 every second.



```
create number myNumber1 = 0
timer every 1 seconds
    myNumber1 add 1
    print to screen myNumber1
```

27. Drag in the Print to Screen **output command** and put it below the variable change, in the yellow box. Select the variable for the Print to Screen. The Print to screen command will display the value of the variable on the screen.

28. Save the code and then run the code. Show children that the value of the variable is increasing by 1 every second and printing to the screen.

29. Have a look at the bottom left-hand corner of the screen as the timer counts. This is the Variable Watch window. It tells us what variables we have in our 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 if necessary.

30. Children should now try to create their own timers in 2Code Gibbon. Remind them to test and save them.

31. Once they have done this, they can try the following Guided lessons that can be accessed on the main 2Code page. Children should watch the videos before each step to help them and use the hints if necessary. To access a hint or view the video again, click on the ?.button top-right and then on the Hint .

- Switching Background - the on/off state of a switch.
- Genie - counting the number of swipes before changing a lamp into a genie. This lesson has a challenge at the end that helps children to explore the variable effects.
- Night and Day Gibbon - the numbers changing in a timer. This lesson has a challenge at the end that helps children to explore the variable effects.

Lesson 5 - Repetition

Aims

- To create a program with an object that repeats actions indefinitely.
- To use a timer to make characters repeat actions.
- To explore the use of the repeat command and how this differs from the timer.

Success criteria

- Children can show how their character repeats an action and explain how they caused it to do so.
- Children are beginning to understand how the use of the timer differs from the repeat command and can experiment with the different methods of repeating blocks of code.
- Children can explain how they made objects repeat actions.

Resources

Unless otherwise stated, all resources can be found on the main unit 3.1 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. 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.
- Example programs and flowcharts. There is a flowchart made using 2Chart and a coded example for each one.
  - Repeat with timer
  - Repeat forever with timer
  - Repeat dance with timer
  - Repeat command with Character
  - Repeat command with turtle
- What does it do? Writing project. Set this as a 2do for the class. You might wish to print the file for some students.
- 2Code Freecode Gibbon (this is found on the main 2Code page).

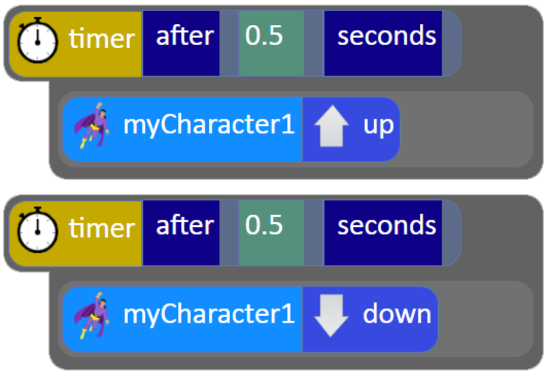
Activities

1. In this lesson, we will be learning some new vocabulary relating to programming. On the board, go through the terms: Sequence, Repeat, Input and Output. You could display these in the classroom and recap them later.

Sequence	This is when a computer program runs commands in order.
Repeat	When a computer program repeats a sequence of commands. In 2Code this could be done using "REPEAT", "REPEAT UNTIL" or using a "Timer" In 2Code a "repeat" command can be used to make a block of commands run a set number of times or to repeat a block of commands forever.
Input	Information going into the computer. An input could be user the moving or clicking the mouse, or the user entering characters on the keyboard. On tablets, there are other forms of input such as finger swipes, touch gestures and tilting the device.
Output	Output is information that comes out of the computer.

	This could be items that appear on the screen or sound that comes out of the speakers. Examples of output are "Print to screen" and "Sound".

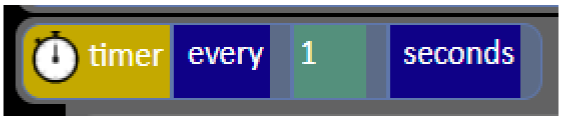
2. Explain that children will be exploring ways to make objects repeat actions today.
3. Firstly, we are looking at making objects repeat actions using a timer. Open the example program Repeat with timer on the whiteboard. The character repeats the action of going up and down twice. Have a look at how this is done in the code and **execute** (run) the code to test it.
4. How could we make the character go up and down forever? Look at the flowchart showing the algorithm for this code. It would be impossible to keep adding the lines …..



```
timer after 0.5 seconds
    myCharacter1 up
timer after 0.5 seconds
    myCharacter1 down
```

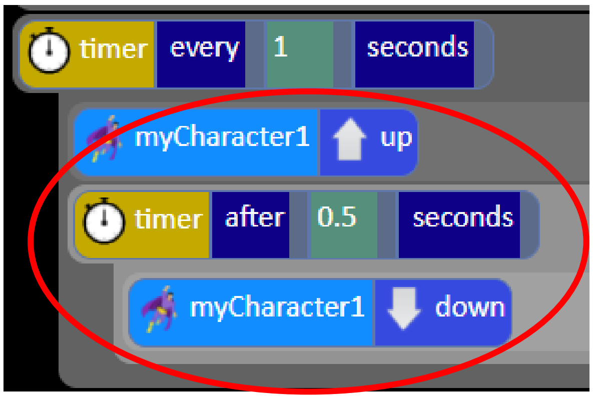
…….inside each other forever.

5. To make the timer repeat forever, you first need to work out how often to repeat the block of code; in the example case, every 1 seconds should be the right amount because he goes up for 0.5 seconds then down for 0.5 seconds then we want him to repeat this again. Look at the flowchart for this example first.
6. Open the example code Repeat forever with timer. Look at the first line of code



```
timer every 1 seconds
```

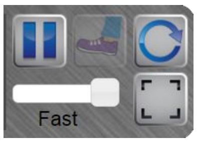
7. Note that the timer says ‘**every**’ rather than ‘**after**’:
8. Inside this timer is the code to repeat:



```
timer every 1 seconds
    myCharacter1 up
    timer after 0.5 seconds
        myCharacter1 down
```

9. Another example of a character repeating actions forever is in the example file Repeat dance with timer. Look at this with the children, can they ‘read’ the code to see how the actions are repeated?

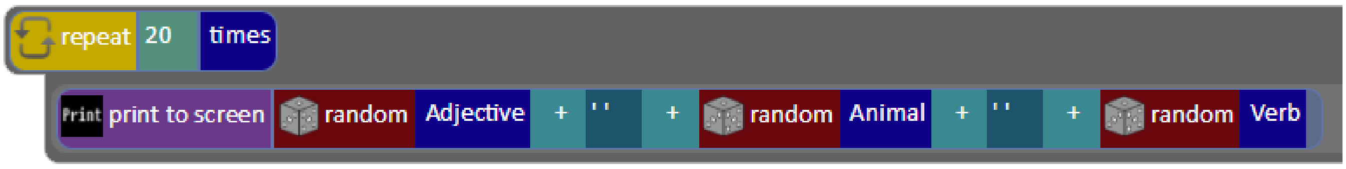
10. Give children time to experiment with making a small program using the timer to repeat actions. They should briefly plan their program by drawing a labelled diagram or a flowchart before coding it.
11. Once they are ready, bring the class back together to discuss the following:
12. Another command that is used to repeat blocks of code is the repeat command. It might seem that you could just use this instead of the timer. However, the repeat command is designed to perform an action (run a block of code) many times as quickly as possible. This is one of the big advantages of using a computer to do tasks; it can be programmed to perform complex calculations much faster than a human can. Therefore, the repeat command might not always be the best choice.
13. Look at the flowchart and example code [Repeat command with character]() It looks like it should work fine.
14. But, when you run it, the code will be performed so fast that you can't actually see the actions! You can try running the code in slow mode and you'll see that the actions do happen.
15. Use the slider bar at the bottom right to alter this before you press 'OK' on the message at the start:



So, in this case, the Repeat command isn't really the best one to use.

16. Lastly, show the children the example of using the repeat command with a turtle object. It does work, just too fast to use in a visual program. An example of this is in [Repeat command with turtle]().
17. . The 'What does it do?' sheet asks children to predict what code will do when run, run the code to check their predictions and then use the code to make a better program. All code snippets use repeat; they are reproduced below. Give children time to complete the sheet (or sections for at least one code snippet). They should then try to adapt the code to make a new program using the same coding blocks and ideas. For example, could they write a program that asks the user for a number and then lists the first 50 numbers in that times table?

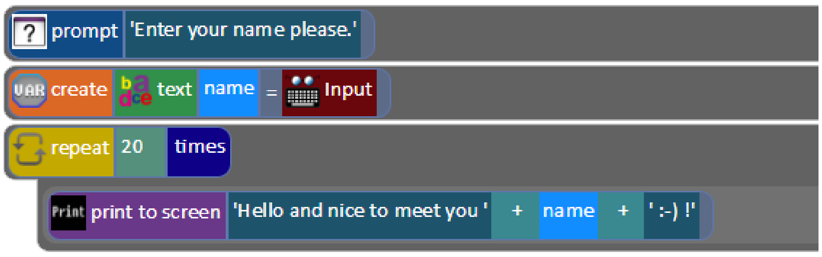
Sentence generator using repeat command.



```
repeat 20 times
    print to screen  random Adjective + ' ' + random Animal + ' ' + random Verb
```

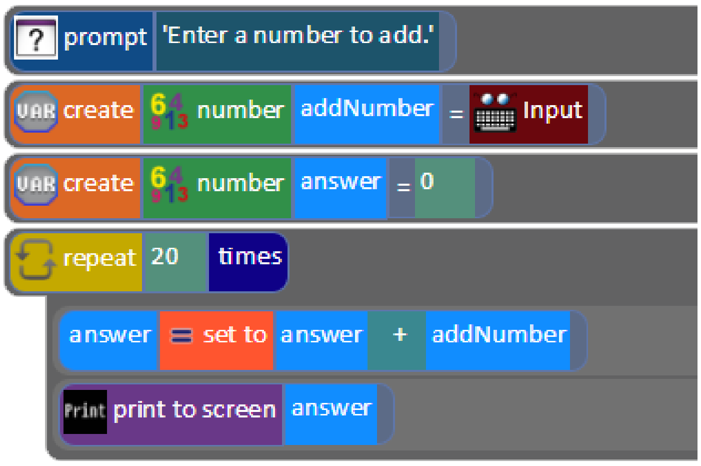
This instantly generates sentences by combining random adjectives, animals and verbs.

Welcome you



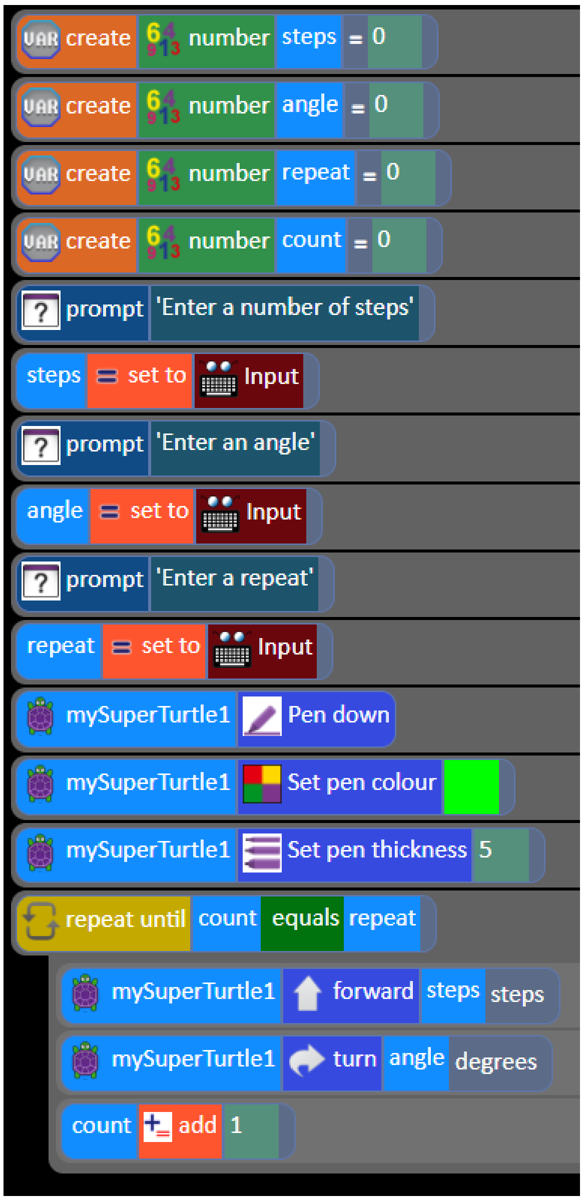
```
prompt 'Enter your name please.'
create text name = Input
repeat 20 times
    print to screen 'Hello and nice to meet you ' + name + ' :-) !'
```

Number sequence maker using user input



```
prompt 'Enter a number to add.'
create number addNumber = Input
create number answer = 0
repeat 20 times
    answer set to answer + addNumber
    print to screen answer
```

Logo Machine (Requires a Turtle object)



```
create number steps = 0
create number angle = 0
create number repeat = 0
create number count = 0
prompt 'Enter a number of steps'
steps set to Input
prompt 'Enter an angle'
angle set to Input
prompt 'Enter a repeat'
repeat set to Input
mySuperTurtle1 Pen down
mySuperTurtle1 Set pen colour 🟩
mySuperTurtle1 Set pen thickness 5
repeat until count equals repeat
    mySuperTurtle1 forward steps steps
    mySuperTurtle1 turn angle degrees
    count add 1
```

Lesson 6 - Debugging

Aims

- To know what debugging means.
- To understand the need to test and debug a program repeatedly.
- To debug simple programs.
- To understand the importance of saving periodically as part of the code development process.

Success criteria

- Children can explain what debug (debugging) means.
- Children have a clear idea of how to use a design document to start debugging a program.
- Children can debug simple programs.
- Children can explain why it is important to save their work after each functioning iteration of the program they are making.

Resources

Unless otherwise stated, all resources can be found on the main unit 3.1 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.
- Debugging Process for display on the board.
- Debug Challenges Gibbon (found near the bottom of the main 2Code page in the Debug Challenge section).
- 2Connect Tool; this is in the Tools section of Purple Mash.

Activities

1. Review the terms below. Children will have done this in year 2.

Bug

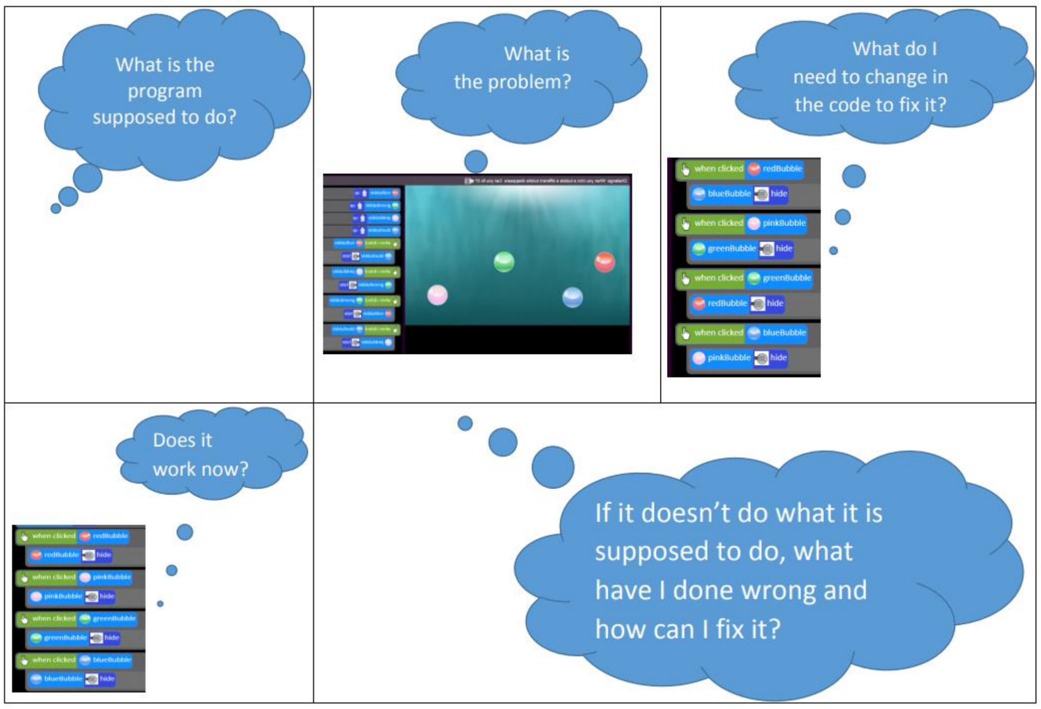
A problem in a computer program that stops it working the way it was designed.

Debug/Debugging

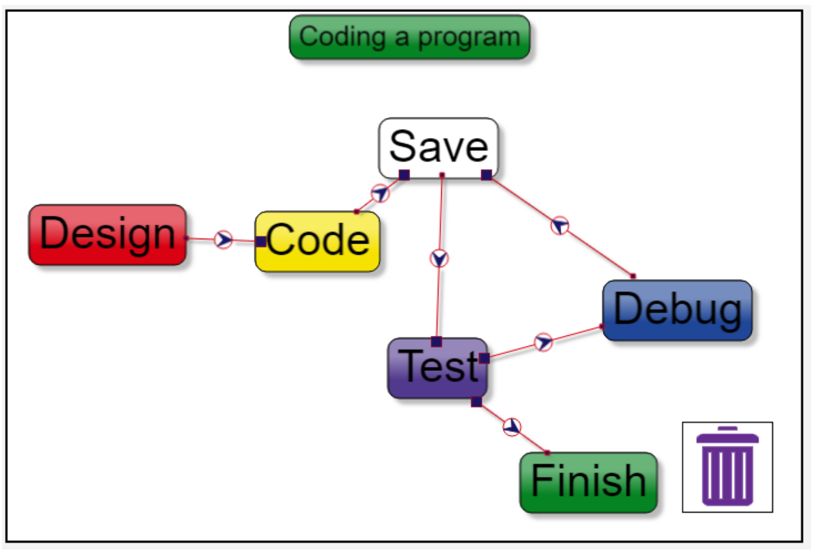
2. Remind children of the debugging steps using the guide:

3. Children should open Debug Challenges Gibbon (found near the bottom of the main 2Code screen in the Debug Challenge section). Children should work through challenges for about 10 minutes.

4. Ask children what they think the stages of creating a computer program are. Record their ideas on a white board or using 2Connect. The aim is to end up with something resembling this example:


What is the program supposed to do?
What is the problem?
What do I need to change in the code to fix it?
Does it work now?
If it doesn't do what it is supposed to do, what have I done wrong and how can I fix it?

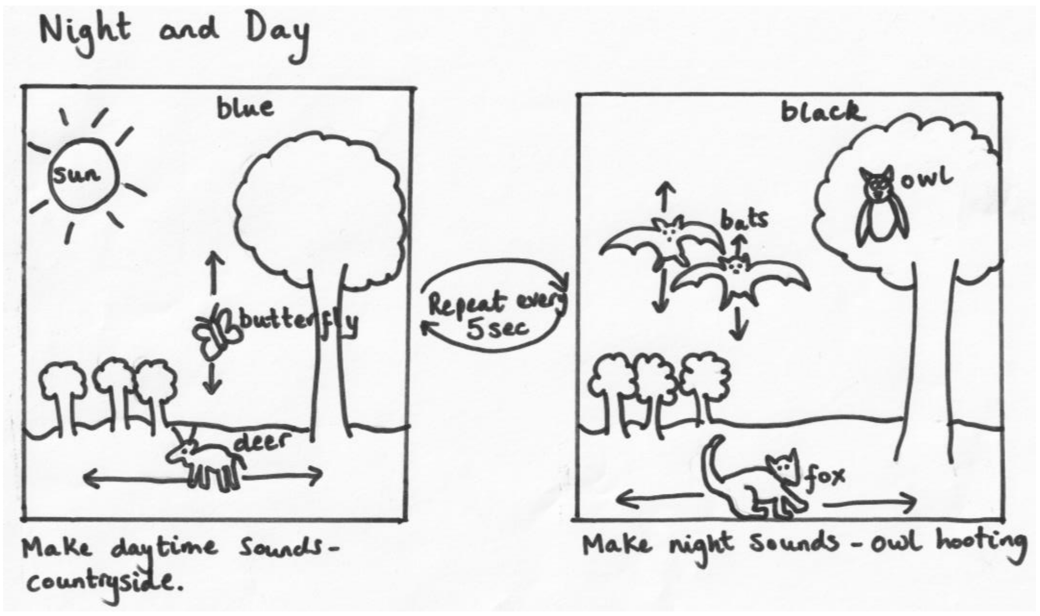
5. Ask them where 'Saving' comes into the process. Should they just save at the end? Why not? Try adding



'Save' in and then adding arrows to show the order of the process.

6. Children will hopefully conclude that you must save regularly so you don't lose any work. It also means that if you mess up your code you can go back to the last version that you saved (by using the open file button). Compare this to other work that they do on the computers, is the process different? Should you always remember to save every so often, whether you are painting a picture, writing a story or making some code?

7. Children will now be using one of their own saved programs that they created in a previous Free Code lesson using Free Code Gibbon. They should decide which one of their programs they want to use, they will be breaking it for another child to debug. They should first write down or draw a design/algorithm diagram to show exactly what the program is supposed to do. Here is an example using a diagram:


Night and Day
blue
sun
butterfly
deer
Repeat every 5sec
black
bats
owl
fox
Make daytime Sounds - countryside.
Make night Sounds - owl hooting

8. Children should then save their work with a different name. Ask them why they think they should do this? Because they are going to be changing the program and they won't want to lose the original version. Giving it a different name means that there are now two versions. Discuss how to name the two versions so they don't get muddled up.

9. They should break a maximum of five pieces of code. Each time they break a bit, they should write down what they broke to stop their program from working.

10. Once they have finished breaking their program, children should save their program using their name and the name of the program in a *class* folder –

11. They should then share the original design with their partner.

12. Each child should open their partner's program from the class folder and each should try to fix it using the debugging steps (display these on the board). Once the child has finished, they should check that the program works according to the explanation that the creator has specified. If it does not work according to the explanation, they must go back and fix the code until it does.

Assessment Guidance

The unit overview for year 3 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 ongoing support, children can turn a real-life situation into an algorithm for a program that has cause and effect (Unit 3.1, lesson 2, step 7) and use their algorithm to write simple two step programs using 2Code (Unit 3.1 Lesson 2, step 8). Furthermore, they can identify an error within a program and fix it (Unit 3.1 Lesson 6). Pupils can design and code a program that follows a simple sequence (lesson 1 and lesson 2). Children experiment with the use of timers to achieve repetition effects in their programs (lesson 5) – they might need adult support to identify the source of unintended effects when using timers. Children can use simple ‘if statements’ to introduce selection to their coding (lesson 3 Children’s designs for their programs, show that they are thinking of the structure of a simple program in logical, achievable steps (lessons 1 and 2). Children can make good attempts to ‘read’ code and predict what will happen in a program which can help them to correct errors (lesson 6).
Expected	Children can turn a simple real-life situation into an algorithm for a program by deconstructing it into manageable parts (Unit 3.1, lesson 2, step 7-8). 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 (lessons 1 & 2). Most children can explain the choice of commands they have included in their program and what they achieve (Unit 3.1. Lesson 1 Point 28). Most children can integrate multimedia components such as sounds, animation and images into their coding. They can apply specific actions to these objects to animate them as part of the overall process of creating their own program (Unit 3.1. Lesson 1). Pupils have a clear idea of how to design and code a program that follows a simple sequence (lessons 1 and 2). Children experiment with the use of timers to achieve repetition effects in their programs – they can determine whether a timer should be called every x seconds or after x seconds and the difference between the two (lesson 5). They are beginning to understand the difference in effect of using a timer command rather than a repeat command when creating repetition effects in their coding (lesson 5). Children can use ‘if’ statements to bring selection into their own coding (unit 3.1, lesson 3). They understand how variables can be used to store information while a program is executing (unit 3.1, lesson 4) and make attempts to use and manipulate the value of variables. Children can ‘read’ others’ code and predict what will happen in a program which helps them to correct errors (lesson 6). They can also make good attempts to fix their own bugs as their coding becomes more complex (lessons 3,4 & 5).

Assessment Guidance	
	They can be reflective on how successful they were at creating their programs and how the previous learning has helped them with the planning aspect of their program (Unit 3.1. Lesson 2 Point 10).
Exceeding	Children are attempting to turn increasingly complex real-life situations into algorithms for a program by deconstructing the situation into manageable parts. Children's design shows that they are thinking of the required task and how to accomplish this in code (Unit 3.1, lesson 2). Children can identify an error within a program that prevents it following the desired algorithm and then fix it (Unit 2.1 Lesson 6). 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 3) and repetition (lesson 5). Children have a good understanding of when to use a timer in a program rather than a 'repeat' command to for repetition (lesson 5) and this is evidenced in their program designs. Children make use of variables in their programs and combine these with timers to creative effect (lesson 4). 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. Children can 'read' others' code and predict what will happen in a program which helps them to correct errors (lesson 6). They exhibit greater ease at fixing their own bugs as their coding becomes more complex. (lessons 3,4 & 5).