



Computing  
Scheme of Work

**CRASH COURSE**



## Coding Unit for Children in Year 3

Who haven't used 2Code in Year 1 or 2



Year Group: 3  
Number of  
Lessons: 6

From **2**simple





## Year 3 Crash Course – Introduction

The crash-course aims to prepare year 3 children for using the Computing Scheme of Work Coding unit in year 4.

### Differentiation

The Gibbon 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.

To enhance children's ability to code and understand the process of coding and design, children should have had as many of the following experiences as possible:

### Challenges

When using the guided activities, children should have attempted the challenges at the end of the guided lessons in 2Code and come up with solutions to these either individually or using shared coding as a group or class.

### Free coding

Children will benefit from spending some time using:

- Y1-2 Free code Chimp (or Free code scenes)
- Y3-4 Free code Gibbon
- Y5-6 Free code Gorilla

To create their own programs.

### Program Design

To master coding skills, children need to have the opportunity to explore program design and put computational thinking into practice. The crash course suggests some designing before coding in the plans. Children could do this through:

- 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               | Covered by guided activities in 2Code  | How to provide additional opportunities   |
|--|-------------------------|--|---|
| <div style="text-align: center;">                     Most scaffolded<br/>  </div>   | Copying code            | ✓  |   |
|  | Targeted tasks          | Read and understand code ✓<br>Remix code to achieve a particular outcome ✓<br>Debugging ✓    | <ul style="list-style-type: none"> <li>• Use printed code snippets so that children can’t run the code but must read it.</li> <li>• Include unplugged activities and ‘explaining’ tasks e.g. ‘how do variables work?’</li> </ul>  |
|  | Shared coding           | Sharing Challenge activities →   | <ul style="list-style-type: none"> <li>• Complete guided activity challenges as a class.</li> <li>• After completing challenges; share methods to create a class version of the challenge.</li> <li>• Free coding as a class</li> </ul>   |
|  | Guided exploration      | Exploring a limited repertoire of commands ✓<br>Remixing code ✓                              | <ul style="list-style-type: none"> <li>• Explore commands in free code before being taught what they do. Use questioning to support children’s learning.</li> </ul>   |
|  | Project design and code | Guided activities ✓<br>Guided challenges at the end of each guided activity ✓<br>Free code ✓ | <p><b>Projects (imitate, innovate, invent, remix)</b></p> <p>There are different ways to scaffold learning in projects. This process can be applied to programming projects;</p> <ul style="list-style-type: none"> <li>• Using example projects.</li> <li>• Create a project that imitates a high-quality exemplar.</li> <li>• Remixing ideas.</li> <li>• Independently creating a brand-new program.</li> </ul> |
| <div style="border: 1px dashed black; padding: 5px; margin: 10px auto; width: 60%;"> <p>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 <i>imitation</i> of an example text. Next, they create a variation of the text (<i>remix and innovate</i>). Finally, they get to <i>inventing</i> a brand-new version.</p> </div> |                         |  |   |
| <div style="text-align: center;">                     Least scaffolded                 </div>  | Tinkering               | Use Free code Gorilla to access the full suite of 2Code objects and commands ✓               | Use Free code to play and explore freely.   |

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



## Year 3 Crash Course – Medium Term Plan

| Lesson   | Aims   | Success Criteria  |
|----------|--|---|
| <u>1</u> | To explain what coding is.<br><br>Introduction to the 2Code interface including the possible actions of character, car and animal objects.<br><br>Tinkering with 2Code | <ul style="list-style-type: none"> <li>Children can explain that coding is how computer programs are created.</li> <li>Children can navigate around the 2Code interface, dragging and dropping code blocks and running code.</li> </ul>   |
| <u>2</u> | To use timers in 2Code to create differing effects.  | <ul style="list-style-type: none"> <li>Children have used timers to create the following effects:                             <ul style="list-style-type: none"> <li>A clock simulation - Tick Tock challenge.</li> <li>To make objects appear to disappear - Magician</li> <li>Making animations to tell stories in sequence – Superheroes, Sparklers and Guard the Castle 2Children can show that their vehicles move at different speeds.</li> </ul> </li> </ul> |
| <u>3</u> | To use repetition commands   | <ul style="list-style-type: none"> <li>Children can show how their character repeats an action and explain how they caused it to do so.</li> <li>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.</li> <li>Children can explain how they made objects repeat actions.</li> </ul>  |
| <u>4</u> | To introduce If statements to allow selection in a program.  | <ul style="list-style-type: none"> <li>Children can create an 'if' statement in their program.</li> <li>Children can use a timer and 'if' statement to respond to the actions of a character and change their actions.</li> </ul>   |
| <u>5</u> | Debugging.   | <ul style="list-style-type: none"> <li>Children can explain what steps to follow to debug a program.</li> <li>Children can explain what they did so that my computer program did not work.</li> <li>Children can explain how they debugged a partner's program.</li> </ul>  |
| <u>6</u> | To introduce variables.  | <ul style="list-style-type: none"> <li>Children can explain what a variable is in programming.</li> <li>Children can explain why variables need to be named.</li> <li>Children can create a variable in a program.</li> <li>Children can set/change the variable values appropriately to create a timer.</li> </ul>   |



# Lesson 1 – Introduction to coding with 2Code

## Aims

- To explain what coding is.
- Introduction to the 2Code interface including the possible actions of character objects.

## Success criteria

- Children can explain which commands they included in their program and what they achieve.
- 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.

- Code block cards. Children will need to use a few copies of each picture to create code away from the computer.
- 2Code Freecode Gibbon (this is found on the [main 2Code page](#)).
- (Optional) Exercise books to be used as 2Code workbooks for recording coding exercises, if desired.
- **Note:** Coding vocabulary is highlighted in bold in this lesson plan, this vocabulary should be used so that children pick up the appropriate coding vocabulary in context.

## Activities

1. Explain to the children that we are going to be coding. Ask them if they know what coding is? Discuss briefly that it is the way that computer programmers input instructions into computers to create programs. Can they give any examples of computer programs that they have used?
2. Start off by doing some activities where the children must follow or give clear instructions.
3. Choose two children; one is a robot and the other is a **coder**. The coder needs to direct the robot to walk from one place in the classroom to another. How can they give the instructions so that the robot does not crash into objects in the way? Repeat a few times in different locations.
4. Tell the children that you are now going to be the robot and they are the coders. Stand by the whiteboard and ask the children to give you clear instructions, one step at a time, for drawing a smiley face.
5. Once you have a set of clear instructions, introduce the term **algorithm**.

A precise step by step set of instructions used to solve a problem or achieve an objective.

The children have created an algorithm to draw a smiley face.

6. Discuss the way that coding languages use symbols rather than whole sentences; they turn the **algorithms** into code. Can they come up with their own symbols for the instructions for drawing a smiley face? For example, holding their fingers in a circle shape for the face and combining this with a symbol for 'small' for the eyes.

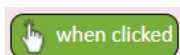


## Purple Mash Computing Scheme of Work – Y3 Coding Crash Course – Lesson 1


These symbols could be written on the whiteboard to create a program. This can become more complex, depending upon the understanding and interest level of the children. Some children will enjoy working out how their 'code' could be adapted to draw a sad or sleeping face. Some children will want to be precise about the placement of the shapes, e.g. a symbol to show putting the smile *below* the eyes.











7. Show the children the printed code block cards. Explain that these are examples of code used on a computer. Can they suggest ways to combine the cards to make instructions?

8. Put [Free Code Gibbon](#) on the board and point out the code blocks on the left-hand side. They will see the



block from the cards but not the bubbles. Explain that this is because we haven't added any bubbles yet; they are called **objects** in 2Code.

9. Click on the  button to go into Design Mode and explain that this is where you set up the look of your program including adding the objects. The different object types in 2Code Gibbon are on the left-hand panel:





|   |   |  |  |   |
|---|---|--|--|---|
| <br>button   | <br>number     | <br>input   | <br>label | <br>shape   |
| <br>vehicle | <br>character | <br>turtle | <br>food | <br>animal |



Background

10. To add an **object**, you drag it onto the screen. Drag a vehicle, a character and an animal into Design View.

11. Explain that the background object is always added. You could click on it to see its **properties**:

| Property   | Value   |
|--|---|
| name   | background  |
|  colour |  |
|  image  |  |
| Grid size  | 4   |

12. Change the background by double-clicking on the question mark, this opens the Clipart picker where you can select an existing background, upload an image from your device or paint an image. Choose any background.



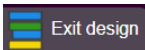
Purple Mash Computing Scheme of Work – Y3 Coding Crash Course – Lesson 1

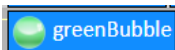
13. Click on the character **object** that you added and change the name **property** to 'greenBubble':

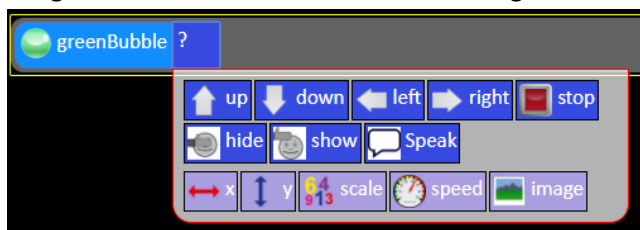
| Property         | Value       |
|------------------|-------------|
| type             | character   |
| name             | greenBubble |
| x                | 14          |
| y                | 8           |
| movement         | Stopped     |
| allow off screen | No          |
| scale            | 100         |
| speed            | 2           |
| image            |             |
| show/hide        | show        |

14. Next double-click on the image **property** to open the clipart picker. You will find the green bubble image in the bubble section of the clipart picker (use the drop-down box at the top). The bubble will probably be too big, so look for the scale **property** and click on it to reduce the scale of the bubble.



15. Show children how to save their work into their work folders and remind them of the need to save regularly with sensible names to avoid losing their work.

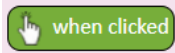
16. Return to Code View  and point out that the added **objects** now appear on the left of the screen below the existing code blocks.

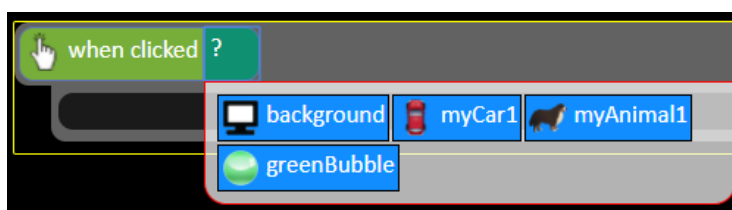
17. Drag a  block into the coding area of the screen; this is the grey bar at the top of the main area.



18. Choose an **action** for it e.g. .

19. Now click on the  button to run the code and you should see the green bubble moving up. Click  to return to the coding view

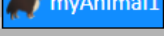


20. Drag a  block below the line of code that you just wrote. This is called an **event** block; when an **object** is clicked it triggers an **event** (the click event) which will have an **action**



21. Look at how this **event** works:





- You select the **object** that will be clicked – select 
- You then drag in the **object** that will perform an **action** when the first object is clicked into the indented part of the code window – emphasise the importance of this to the children - select 
- You then select an **action** for that object – select 

22. What do children think will happen when the code is run now? Save the code and see if they predicted correctly; the green bubble will move up, when the dog is clicked on the bubble will move down.

23. We need to say what happens when we click on the object. This is called ‘output’.

24. The children will have been listening for a long time at this point, so this is now their time to have a **tinker** with 2Code and see what they can discover. This tinkering could be completely free, or you could give some directives as suggested below. Remind children to save their work regularly and use a different name for each program that they make.

- Add different types of objects and see how they have different actions e.g. compare character, vehicle and turtle objects – [see Appendix 2](#) for details of what these are.
- Work out what each of the Commands does (the code blocks at the top above the objects) – [see Appendix 3](#) for details.
- Try adding sound when objects are clicked



- Try to work out how a timer or a repeat block works

25. If the children have workbooks, they can print their code and write about what they have discovered and what they would like to try doing next.



## Lesson 2 – Using timers in 2Code

### Aims

- To use timers in 2Code to create differing effects.
- To use appropriate coding vocabulary including Object, Action, Output, Control and Event.

### Success criteria

- Children have used timers to create the following effects:
  - A clock simulation - Tick Tock challenge.
  - To make objects appear to disappear - Magician
  - Making animations to tell stories in sequence – Superheroes, Sparklers and Guard the Castle 2

### 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](#)).
- Vocabulary 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.
- Exercise books to be used as 2Code workbooks for recording coding exercises, if desired.
- Headphones so that children can hear the help videos and instructions on their devices.

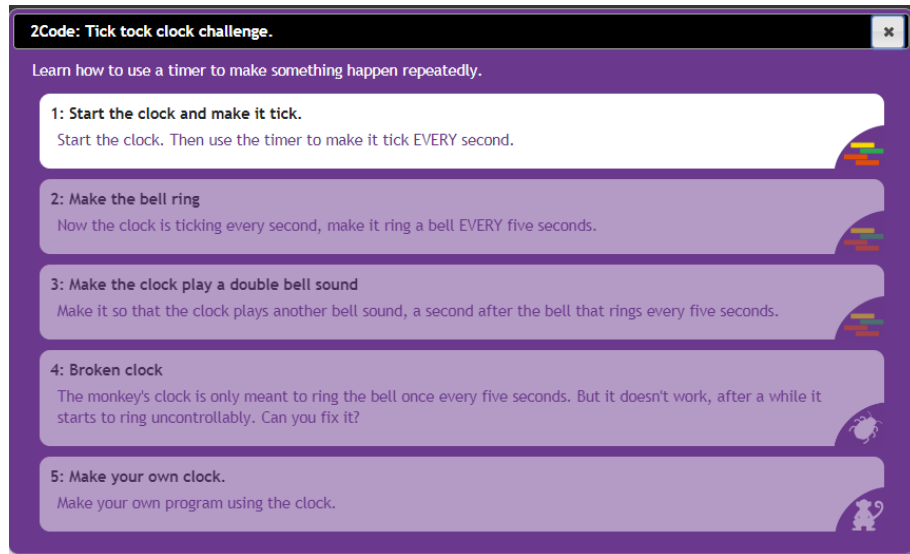
### Activities

1. Remind children that last lesson they were introduced to some coding terms. Who can remember any?
2. Review Algorithm, Object, Properties, Event, and Action using the vocabulary cards.
3. Today children will be using a variety of the guided coding activities to explore the use of timers in 2Code. They will also be using sound effects, show/hide actions and using angle properties of objects.
4. Show children where to find the Guided activities in the 2Code area. They will find today's activities in the Chimp section. You could set these activities as 2dos to make it easier for the children to find them.




Purple Mash Computing Scheme of Work – Y3 Coding Crash Course – Lesson 2


5. Open the Tick Tock challenge activity on the whiteboard. Explain that you are going to do this activity as a class and everybody can help:



In this activity there are 5 steps. The symbols on the right-hand side of each activity show what type of activity it is. The first 3 steps are coding activities, step 4 is a debugging activity (this means fixing some code) and the last step is a challenge.

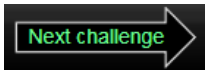
6. Click on step 1 and watch the video together. Because the children have not done much coding before, the task will not be immediately obvious but reassure the children that the activity is set up to help them. There is a Hint button on the bottom left which gives extra clues but don't open this for now. Click OK.
7. 2Code opens with just a limited number of objects and commands. You can look at the design view to see the clock then return to the code view. At the top of the screen are the instructions (there is a speaker symbol to listen to them read to you).

**Challenge: Start the clock. Then use the timer to make it tick EVERY second.** 

8. You can also watch the video again by clicking on the  at the top right.
9. Can anyone suggest how to start the clock? (Drag the clock control into the code window for a clue).
10. How could you add a timer? (Drag the timer command into the code window below the clock).
11. The clock needs to tick **every** 1 second so make sure that it does not say **after** (click on after to change this).
12. What action should happen every 1 second? (The sound tick should be make – drag the sound into the timer code).



13. When you are happy with the code, click the play button to run it. If you are correct the



link will appear at the top of the screen. Clicking this will take you to the video for step 2.



14. Watch this video and then click on the



if they are really stuck. Watching a hint will affect children's scores but they should be used by children when needed especially as they have not done much prior coding.

15. Complete the rest of the steps as a class including the challenge step. Children tend to rush this step to move onto the next challenge, but this step is really important for them to develop genuine coding skills that they can apply to their own code.

**Note:** the video for the debugging step where the monkey is stuck can be watched by clicking on the



button.

16. Children should then spend the rest of the lesson working on the following guided activities that have been selected to give children a breadth of examples of timer used in 2Code and introduce coding designs that they will not yet have encountered. Remind children to save their work regularly and use a different name for each program that they make.

### Magician

**2Code: Make some magic**

Magic is all about timing. These challenges will teach you how to use the timer tool to help our magician perform magic tricks.

- 1: Make the rabbit disappear**  
The magician wants the rabbit to disappear after five seconds. Use a timer to make this happen.
- 2: Rabbit reappears**  
Now make the rabbit reappear five seconds after it disappeared,
- 3: Magician walks off the stage**  
After the amazing rabbit trick make the magician leave the stage five seconds after the rabbit reappeared
- 4: The rabbit disappears too quickly.**  
The monkey is trying to write a program to make the rabbit disappear after five seconds. Can you fix it?
- 5: Make your own magic**  
Make your own magic sequence for the magician.



## Superheroes

**2Code: Superheroes**

Let's play superheroes

- 1: Make your superhero fly**  
Make the superhero fly up into the air when clicked.
- 2: Make your second superhero disappear**  
Make the second superhero disappear when clicked and then reappear after 1 second.
- 3: From zero to hero**  
When you click the new superhero, increase his scale by 50.
- 4: Broken superheroes**  
Can you unscramble the instructions? Star should fly up, lightning should disappear and reappear and Zero should increase scale by 50
- 5: Make your own superhero sequence**  
Make your own awesome superhero sequence.

## Sparklers

**2Code: Sparklers**

Remember, remember, the 5th of November!

- 1: Make the Sparkler Spin**  
Make a timer that gets called every quarter second. Inside the timer, set the sparkler's rotation to a random value between 1 and 360.
- 2: Let's Jitter**  
Inside the timer, set the sparkler's scale to a random number between 10 and 20.
- 3: Make your own Fireworks Display**  
Make an exciting fireworks display



## Lesson 3 Repetition Commands

### 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.
- 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](#)
- [Worksheet 'What does it do?'](#). You could set this as a 2Do or print in colour.
- 2Code Freecode **Gibbon** tool.

### 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.

**Sequence** - When a computer program repeats a sequence of commands. In 2Code this could be done using "REPEAT", "REPEAT UNTIL" or using a "Timer"

**Repeat** – 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 the 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".

- 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.
- 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.

How could we make the character go up and down forever? It would be impossible to keep adding the lines .....



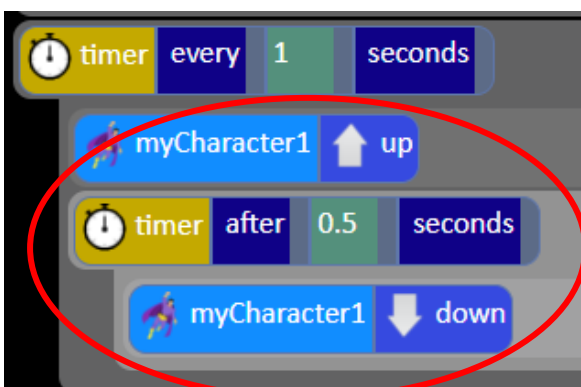
.....inside each other forever.

- 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.
- Open the example [Repeat forever with timer](#). Look at the first line of code



Note that the timer says '**every**' rather than '**after**':

Inside this timer is the code to repeat:




Need more support? [Contact us](#)

Tel: 0208 203 1781 | Email: [sow@2simple.com](mailto:sow@2simple.com) | Twitter: [@2simplesoftware](https://twitter.com/2simplesoftware)



- 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?
- 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 storyboard before coding it.
- Once they are ready, bring the class back together to discuss the following:

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.

- Look at the example code [Repeat command with Character](#). 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.

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.

- If the children have done any work with Turtle objects and Logo, you might want to show them the additional example of using the repeat command in the way that they would 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](#).
- 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 here:

Sentence generator using repeat command.

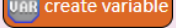


This instantly generates sentences by combining random adjectives, animals and verbs.





Welcome you

To recreate this example, children will need to use the  block. For the time being, do not focus on exactly what a variable is (they will cover this in lesson 4 as it would be too much to squeeze in here). You might want to walk them through the process of creating a variable like the example, this is detailed in [Appendix 4](#) where the process for combining text and variables in 'print to screen' responses is also covered.

Number sequence maker using user input

11. Give children time to experiment with programs using the repeat command to repeat actions. Could they write a program that asks the user for a number and then lists the first 50 numbers in that times table? Remind children to save their work regularly and use a different name for each program that they make.



# Lesson 4 – 'If' Statements

## Aims

- To create a program that responds to the 'if' command or the 'if/else' command
- To use selection within a program.

## Success criteria

- Children can create an 'if' statement in their program.
- Children can create an 'if/else' statement in their program.
- Children can use a timer and 'if' statement to respond to the actions of a character and change their 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.
- 2Code guided activity Guard the Castle (Gibbon) from the [main 2Code page](#).
- 2Code Freecode Gibbon (this is found on the [main 2Code page](#)).
- Blank printed storyboards for designing: [Storyboard - 4 boxes](#) or [Storyboard - 6 boxes](#)
- [Program Design Examples](#) as a reference to possible design style examples.

## Activities

1. Review the vocabulary that children have been introduced to so far, specifically, this includes:

- algorithm
- object
- property
- event
- action
- timer
- repeat

Though they will be familiar with many other terms now such as 'block', 'coder', 'bug' etc

Explain that so far, the children's programs have all followed a **sequence** of events and some have included **repetition** (using repeats and/or timers). Show the **Sequence** vocabulary card. Now we are going to introduce some **selection** into their coding.

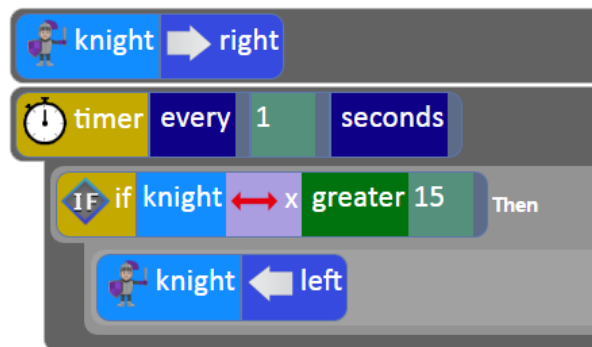
2. Show the **selection** vocabulary card. Then introduce the new vocabulary that they will be learning about today: **'if' statements** using the vocabulary card. Compose some 'if statements' as a class e.g. 'if you listen well then I will be happy', 'if there is rain then it will be wet play' etc.



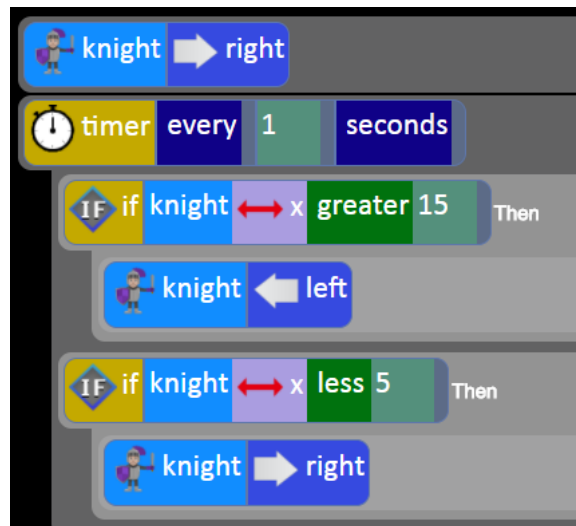
- Open the activity 'Guard the Castle 2' in the **Gibbon** activities and do Step 1 together.

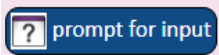


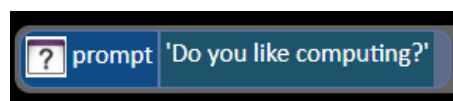
- In Step 2, you must create a timer which checks the X position of the knight every second; **if** the knight's position is greater than 15, **then** he should change direction. Switch to design view and point out the x and y positions displayed in the bottom left corner. Drag the knight around and see how they change. This is to help the children gain an understanding of how measuring the X position can decide whether the knight should change direction.



- 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.




- Now open free code Gibbon and drag a  block into the coding window. Type a question into the input box such as 'Do you like computing?'





7. Drag in an  as follows:

8. Run the code and answer 'yes' then run again and answer 'no'; nothing happens. Discuss that this is not a good response. How could we change it so the computer does something if the person enters 'no'. If

children do not notice the , point it out to them. Show them the definition of if/else using the vocabulary cards.

9. Replace the  with :

10. Run the program again and on the third time enter gibberish to the question. The answer won't be appropriate, what could we do? Here is a possible solution; talk this through with the children.



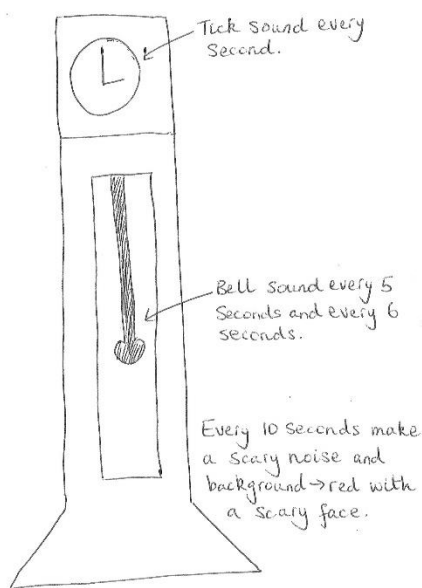
- 11. Children should **plan** a program that uses 'if' or 'if/else' blocks. They could use user input or a more visual program like Guard the Castle. They could even make a simple chat robot using a character object who speaks.



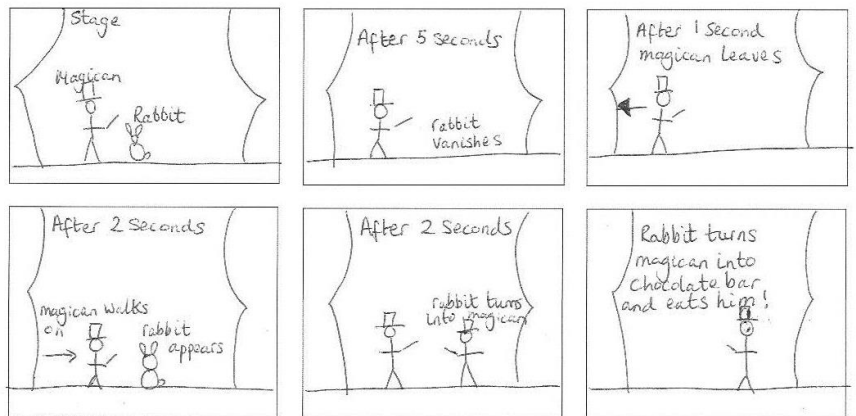
**A note about design:** encourage children to think through their designs and annotate them including their confidence in coding what they have designed (red, amber, green), this will give you feedback on areas that children need help with and help to ensure that children create realistic designs and successful programs for their skill level. Use the [Program Design Examples](#) as a reference to possible design style examples.

- 12. Remind children to save their work regularly and use a different name for each program that they make.

Task: To make a ticking clock with sound effects using a timer



Task: Make an animated story about a magician and a rabbit. Use timers.





# Lesson 5 - Debugging

## Aims

- To understand what debugging means.
- To intentionally break my program.
- To debug my partner's program.

## Success criteria

- Children can explain what steps to follow to debug a program.
- Children can explain what they did so that my computer program did not work.
- Children can explain how they debugged a partner's 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.

- Vocabulary flash cards.
- [Debugging Process](#) for display on the board.
- Debug Challenges Chimp (found near the bottom of the [main 2Code page](#) in the Debug Challenge section).
- Debug Challenges Gibbon (found near the bottom of the [main 2Code page](#) in the Debug Challenge section).
- 

## Activities

1. Review the term 'debug' using the flash cards. What is a bug in a computer program, and what is fixing the bug called? (Debugging.)
2. Before you start to debug a program, you need to think through some steps. Put the debugging steps pdf on the board and talk through the steps.

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?

3. Children should open Debug Challenges Chimp (found near the bottom of the main 2Code screen in the Debug Challenge section). Children should work through challenges for 5–10 minutes. They could also try the Gibbon Debug Challenges.
4. Children will now be using one of their own saved programs that they created in a previous lesson.
5. Children should decide which one of their programs they want to use and, using the first three steps, break their code. But before they make a start, make sure they have written in their workbooks:
  - what their program is supposed to do
  - what the problems are, and
  - what they broke to stop their program from working.
6. They should then write an instruction for their partner, similar to the ones seen in the Debug Challenges, to help them debug their program.
7. They should break a maximum of five pieces of code. 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 –
8. **Do not overwrite the original program!**
9. Each child should open their partner's program from the class folder and each should try and fix the other's program using the debugging steps that are on the board. Once the child has finished, they should check that the program works according to the explanation that the creator has specified in their workbook. If it does not work according to the explanation, they have to go back and fix the code until it does.



# Lesson 6 - Introducing Variables

## Aims

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

- Vocabulary flash cards.
- 2Code Freecode Gibbon (this is found on the [main 2Code page](#)).

## Activities

1. Today we will be working with variables. Use the vocabulary flash card to give a definition of the word
2. Explain the following:

Variables are like boxes in which the computer can store one piece of information.

To find the information from the right box, each box should be labelled. This reserves some space in the computer memory for this variable.


Therefore, each variable (each of our boxes) needs to have a name. The name should be something that helps you remember what it is.

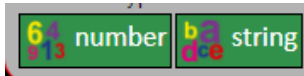
The information inside the box is called the **Variable Value**. The user or the program can change this Variable Value.

3. In 2Code, variables can either be numbers or text (words, phrases or even whole sentences).
4. Open Free Code Gibbon on the IWB and look at the orange variable buttons in the menu on the left-hand side.



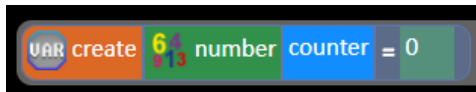


5. Drag a  block into the code window. The drop-down menu gives two options;




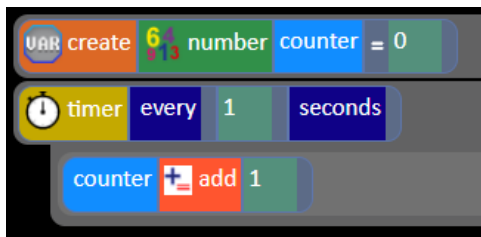
6. Choose 'Number'. Give the variable the name 'counter'. Ask children why we are naming the variable?

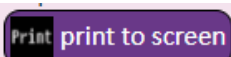
7. We can either define the variable as a specific number or set it to Random. For this example, we will set the variable to 0 as the counter will start from 0.

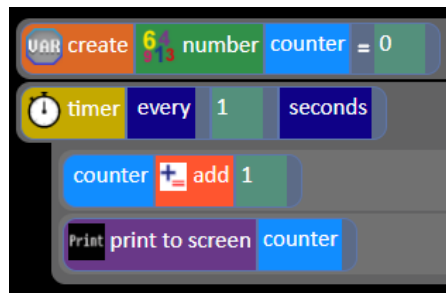


8. Drag in a timer and change 'After' to 'Every'

9. Drag a  into the 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 1 every second.



10. Drag a  block into the timer as well and select your variable.



11. Run the code and show children that it is printing every number that it changes

12. 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.

13. Here are some examples of variables in use in the guided activities, you could look at these as a class or ask children to try the activities:

- the on/off state of a switch (see [Switching Background lesson](#))
- counting the number of swipes before changing a lamp into a genie (see [Genie lesson](#))



- the numbers changing in a timer (see [Night and Day Gibbon lesson](#)).
14. Send children to computers to make their own timers. Once they have a working timer, they can experiment with number variables and see what they can make their programs do.



## Appendix 1: Other Features of 2Code

### Real Code Mode:





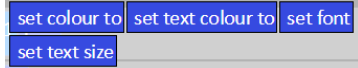

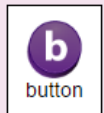









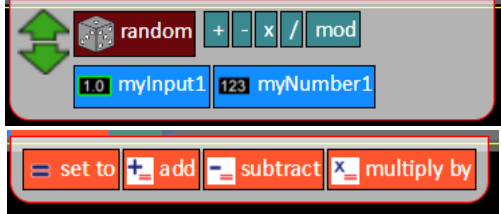





In the more advanced lessons and Free Code modes, it is possible to click the Real Code button and see the code in a simple subset of JavaScript. The code can be edited in Real Code mode, and clicking the Edit Blocks button will bring the user back to the usual graphical representation. If the user types code into the Real Code window that is syntactically incorrect, the Real Code window will flash red. Changing back to Edit Blocks will restore the code to the last state that was syntactically valid.

### Sharing:

Once a 2Code program has been saved into Purple Mash, it can be shared by clicking on the globe icon in the toolbar. This will display a window with a hyperlink and an embed code. Clicking the hyperlink will launch the 2Code program in a web browser in full-screen mode and the embed code can be used to embed a 2Code program into a blog or a website. A Purple Mash login is not required to run a shared 2Code program.




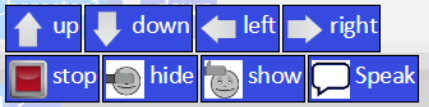

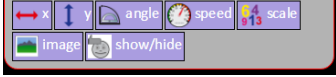
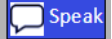







## Appendix 2: Actions for Gibbon objects

| Object  | Properties in Design View   | Properties in Code View | Actions in code view |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
|---|---|-------------------------|----------------------|------|------------|------|------------|--------|---|-----------|----|-------------|---|---|---|-------------|---|------------|---|------|--|
|    | <table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>type</td> <td>background</td> </tr> <tr> <td>name</td> <td>background</td> </tr> <tr> <td>colour</td> <td></td> </tr> <tr> <td>image</td> <td>?</td> </tr> <tr> <td>Grid size</td> <td>4</td> </tr> </tbody> </table>   | Property                | Value                | type | background | name | background | colour |  | image     | ?  | Grid size   | 4   |  |  <p>These set the background colour and the properties of any text that is printed to the screen.</p> |             |   |            |   |      |  |
| Property  | Value   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| type  | background  |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| name  | background  |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| colour  |    |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| image   | ?   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| Grid size   | 4   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
|    | <table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>type</td> <td>button</td> </tr> <tr> <td>name</td> <td>myButton1</td> </tr> <tr> <td>x</td> <td>3.825</td> </tr> <tr> <td>y</td> <td>5</td> </tr> <tr> <td>text</td> <td>myButton1</td> </tr> <tr> <td>text size</td> <td>16</td> </tr> <tr> <td>text colour</td> <td></td> </tr> <tr> <td>background</td> <td></td> </tr> </tbody> </table> | Property                | Value                | type | button     | name | myButton1  | x      | 3.825   | y         | 5  | text        | myButton1   | text size   | 16  | text colour |  | background |  | None | None   |
| Property  | Value   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| type  | button  |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| name  | myButton1   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| x   | 3.825   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| y   | 5   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| text  | myButton1   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| text size   | 16  |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| text colour   |    |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| background  |    |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
|  | <table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>type</td> <td>number</td> </tr> <tr> <td>name</td> <td>myNumber1</td> </tr> <tr> <td>value</td> <td>0</td> </tr> <tr> <td>text size</td> <td>18</td> </tr> <tr> <td>text colour</td> <td></td> </tr> <tr> <td>border</td> <td>No</td> </tr> <tr> <td>x</td> <td>8.775</td> </tr> <tr> <td>y</td> <td>3.8</td> </tr> </tbody> </table>   | Property                | Value                | type | number     | name | myNumber1  | value  | 0   | text size | 18 | text colour |  | border  | No  | x           | 8.775   | y          | 3.8   |      |  <p>This displays a number on the screen which can be set to different values and/or have calculations performed on it. In the image above, myInput1 and myNumber1 are objects that also have a number value and the value of these can be set to affect the value of the number object.</p> |
| Property  | Value   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| type  | number  |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| name  | myNumber1   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| value   | 0   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| text size   | 18  |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| text colour   |    |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| border  | No  |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| x   | 8.775   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| y   | 3.8   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
|  | <table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>type</td> <td>number</td> </tr> <tr> <td>name</td> <td>myNumber1</td> </tr> <tr> <td>value</td> <td>0</td> </tr> <tr> <td>text size</td> <td>18</td> </tr> <tr> <td>text colour</td> <td></td> </tr> <tr> <td>border</td> <td>No</td> </tr> <tr> <td>x</td> <td>8.775</td> </tr> <tr> <td>y</td> <td>3.8</td> </tr> </tbody> </table>   | Property                | Value                | type | number     | name | myNumber1  | value  | 0   | text size | 18 | text colour |  | border  | No  | x           | 8.775   | y          | 3.8   |      | <p>This displays an input box on the screen into which a number can be typed.</p> <p>In the code, input can be set to different values and/or have calculations performed on it.</p>   |
| Property  | Value   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| type  | number  |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| name  | myNumber1   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| value   | 0   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| text size   | 18  |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| text colour   |    |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| border  | No  |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| x   | 8.775   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |
| y   | 3.8   |                         |                      |      |            |      |            |        |   |           |    |             |   |   |   |             |   |            |   |      |  |




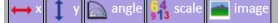




Purple Mash Computing Scheme of Work – Y3 Coding Crash Course – Appendix 2

| <br>label   | <table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr><td>type</td><td>label</td></tr> <tr><td>name</td><td>myLabel1</td></tr> <tr><td>text</td><td>My Label</td></tr> <tr><td>background</td><td></td></tr> <tr><td>text size</td><td>26</td></tr> <tr><td>text colour</td><td></td></tr> <tr><td>border</td><td>No</td></tr> <tr><td>x</td><td>19.875</td></tr> <tr><td>y</td><td>5.4</td></tr> </tbody> </table>   | Property | Value | type | label     | name | myLabel1     | text | My Label | background |        | text size        | 26      | text colour      |                | border | No  | x      | 19.875 | y     | 5.4 | None   | None. The text for the label is set in design view and cannot be changed   |   |  |  |   |
|--|---|----------|-------|------|-----------|------|--------------|------|----------|------------|--------|------------------|---------|------------------|----------------|--------|-----|--------|--------|-------|-----|--|--|---|--|--|---|
| Property   | Value   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| type   | label   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| name   | myLabel1  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| text   | My Label  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| background   |   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| text size  | 26  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| text colour  |   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| border   | No  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| x  | 19.875  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| y  | 5.4   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| <br>shape   | <table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr><td>type</td><td>shape</td></tr> <tr><td>name</td><td>myShape1</td></tr> <tr><td>x</td><td>21.175</td></tr> <tr><td>y</td><td>11.475</td></tr> <tr><td>speed</td><td>0</td></tr> <tr><td>size</td><td>3</td></tr> <tr><td>sides</td><td>3</td></tr> <tr><td>colour</td><td></td></tr> <tr><td>angle</td><td>0</td></tr> </tbody> </table>   | Property | Value | type | shape     | name | myShape1     | x    | 21.175   | y          | 11.475 | speed            | 0       | size             | 3              | sides  | 3   | colour |        | angle | 0   |  <p>The options offered will depend upon the property selected.</p> |  <p>These make the object move in different directions.<br/>Stop, hide or show the object.<br/>Make the object speak by displaying a speech bubble.</p> |   |  |  |   |
| Property   | Value   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| type   | shape   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| name   | myShape1  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| x  | 21.175  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| y  | 11.475  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| speed  | 0   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| size   | 3   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| sides  | 3   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| colour   |   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| angle  | 0   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| <br>vehicle   | <table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr><td>type</td><td>vehicle</td></tr> <tr><td>name</td><td>myVehicle1</td></tr> <tr><td>x</td><td>5.175</td></tr> <tr><td>y</td><td>15.625</td></tr> <tr><td>allow off screen</td><td>No</td></tr> <tr><td>rotation style</td><td>Face the angle</td></tr> <tr><td>angle</td><td>0</td></tr> <tr><td>speed</td><td>0</td></tr> <tr><td>scale</td><td>100</td></tr> <tr><td>image</td><td></td></tr> <tr><td>show/hide</td><td>show</td></tr> </tbody> </table> | Property | Value | type | vehicle   | name | myVehicle1   | x    | 5.175    | y          | 15.625 | allow off screen | No      | rotation style   | Face the angle | angle  | 0   | speed  | 0      | scale | 100 | image  |  | show/hide   | show   |  <p>The options offered will depend upon the property selected.</p> |  |
| Property   | Value   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| type   | vehicle   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| name   | myVehicle1  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| x  | 5.175   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| y  | 15.625  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| allow off screen   | No  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| rotation style   | Face the angle  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| angle  | 0   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| speed  | 0   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| scale  | 100   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| image  |   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| show/hide  | show  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| <br>animal<br><br><br>character<br><br><br>food | <table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr><td>type</td><td>character</td></tr> <tr><td>name</td><td>myCharacter1</td></tr> <tr><td>x</td><td>32.925</td></tr> <tr><td>y</td><td>4.175</td></tr> <tr><td>movement</td><td>Stopped</td></tr> <tr><td>allow off screen</td><td>No</td></tr> <tr><td>scale</td><td>100</td></tr> <tr><td>speed</td><td>2</td></tr> <tr><td>image</td><td></td></tr> <tr><td>show/hide</td><td>show</td></tr> </tbody> </table>  | Property | Value | type | character | name | myCharacter1 | x    | 32.925   | y          | 4.175  | movement         | Stopped | allow off screen | No             | scale  | 100 | speed  | 2      | image |     | show/hide  | show   |  |  <p>These make the object move in different directions.<br/>Stop, hide or show the object.<br/>Make the object speak by displaying a speech bubble.</p> |  |   |
| Property   | Value   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| type   | character   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| name   | myCharacter1  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| x  | 32.925  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| y  | 4.175   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| movement   | Stopped   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| allow off screen   | No  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| scale  | 100   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| speed  | 2   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| image  |   |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |
| show/hide  | show  |          |       |      |           |      |              |      |          |            |        |                  |         |                  |                |        |     |        |        |       |     |  |  |   |  |  |   |



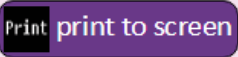
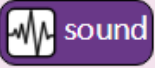

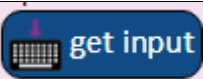






Purple Mash Computing Scheme of Work – Y3 Coding Crash Course – Appendix 2

| <br>turtle                       | <table border="1"><thead><tr><th>Property</th><th>Value</th></tr></thead><tbody><tr><td>type</td><td>turtle</td></tr><tr><td>name</td><td>myTurtle1</td></tr><tr><td>x</td><td>10.825</td></tr><tr><td>y</td><td>15.95</td></tr><tr><td>angle</td><td>0</td></tr><tr><td>scale</td><td>100</td></tr><tr><td>image</td><td></td></tr><tr><td>show/hide</td><td>show</td></tr></tbody></table> | Property | Value | type | turtle | name | myTurtle1 | x | 10.825 | y | 15.95 | angle | 0 | scale | 100 | image |  | show/hide | show |  |  |
|---|---|----------|-------|------|--------|------|-----------|---|--------|---|-------|-------|---|-------|-----|-------|---|-----------|------|---|--|
| Property  | Value   |          |       |      |        |      |           |   |        |   |       |       |   |       |     |       |   |           |      |   |  |
| type  | turtle  |          |       |      |        |      |           |   |        |   |       |       |   |       |     |       |   |           |      |   |  |
| name  | myTurtle1   |          |       |      |        |      |           |   |        |   |       |       |   |       |     |       |   |           |      |   |  |
| x   | 10.825  |          |       |      |        |      |           |   |        |   |       |       |   |       |     |       |   |           |      |   |  |
| y   | 15.95   |          |       |      |        |      |           |   |        |   |       |       |   |       |     |       |   |           |      |   |  |
| angle   | 0   |          |       |      |        |      |           |   |        |   |       |       |   |       |     |       |   |           |      |   |  |
| scale   | 100   |          |       |      |        |      |           |   |        |   |       |       |   |       |     |       |   |           |      |   |  |
| image   |    |          |       |      |        |      |           |   |        |   |       |       |   |       |     |       |   |           |      |   |  |
| show/hide   | show  |          |       |      |        |      |           |   |        |   |       |       |   |       |     |       |   |           |      |   |  |
| <p>A turtle moves in a similar way to a floor turtle using Logo type actions. Turn is by a number of degrees.</p> |   |          |       |      |        |      |           |   |        |   |       |       |   |       |     |       |   |           |      |   |  |





## Appendix 3: Commands for Gibbon objects

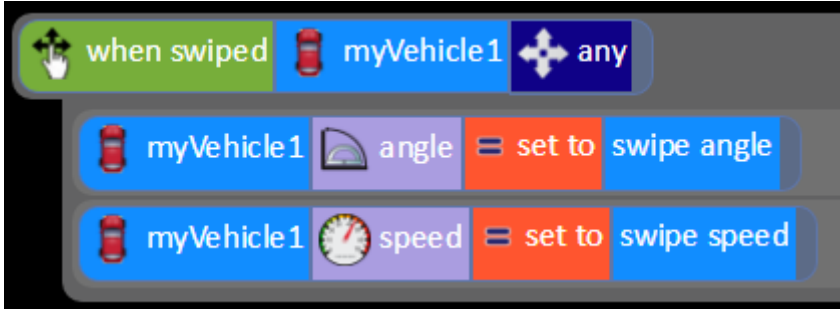

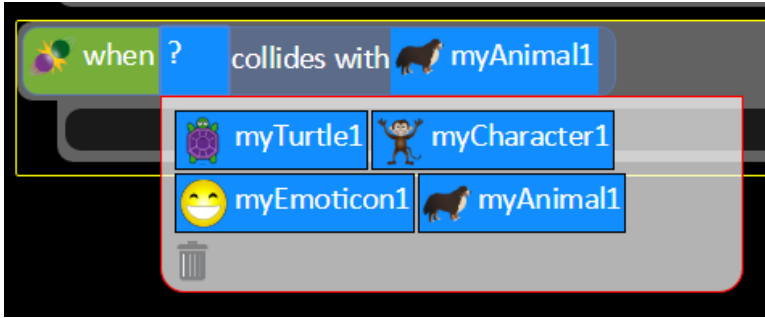
| Command   | Explanation  |
|---|--|
|    | Prints some text specified by the coder to the screen.   |
|    | Causes a sound to play. the sound picker will open for the coder to select a sound, when this code block is added to the code window.  |
|    | Creates a pop-up window with a message for the user and an OK button to click.   |
|    | This command will put a cursor in the top left of the screen and get the input typed onto the screen. For example if you have an alert that asks the user to type their name, you can use this to print their name back to them:<br><br> |
|  | This combines the alert and get input functions, a pop-up screen will ask the user to enter something and they type it into a text box on the pop-up screen.   |
|  | Create a timer. The coder can select whether this time should run after a certain length of time or every x length of time. The time length is measured in seconds or quarter seconds.<br><br>   |
|  | This runs the code inside if a certain condition is met. The condition could depend upon something entered by the user or upon the value of a variable.  |
|  | This runs the code inside the first block if a certain condition is met, otherwise it runs the code inside the second block.   |



|  |   |  |
|--|---|--|
|  |   |  |
|  | <p>Repeats the code inside it either forever or every x (or quarter seconds).</p>   |  |
|  | <p>This command repeats the code inside until a certain condition is met. The condition could depend upon something entered by the user or upon the value of a variable.</p>  |  |
|  | <p>This will restart the program from the beginning. Useful if you want to include a restart button in your program.</p>  |  |
|  | <p>This will launch another 2Code program or open a web page. The following screen will allow the coder to select which. You might want buttons in your program to link to other programs that you have made or to the Internet. The launch command can be useful when you write much bigger programs as you can split them into smaller chunks that launch each other.</p>                             |  |
|  | <p>Runs the code inside it when the specified key is pressed. The code chooses which key (including arrow keys and space bar)</p>   |  |
|  | <p>Runs the code inside it when the object is clicked. The coder is given a choice of all the available objects.</p>  |  |
|  | <p>Useful for tablets. This command runs the code inside it when an object is swiped. The coder chooses the object and the direction of the swipe. You can use the swipe speed and swipe angle in the code. This works especially well when applied to objects that can have both their angle and speed set, such as vehicles. Remember that you can change the image of a vehicle to anything else</p> |  |



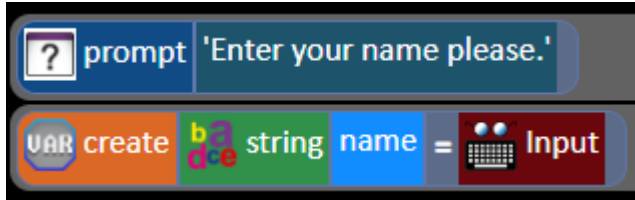


|   |  |
|---|--|
|   | <p>such as a person if you want to be able to do this with objects that don't look like vehicles.</p>  |
|  | <p>Runs the code inside it when two objects collide. The coder selects the two objects.</p>           |



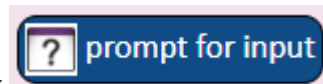
## Appendix 4 – Creating variables

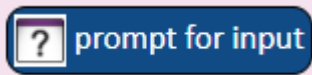
If children are trying to recreate the examples in lesson 2, they will come across the lines:



They will not have encountered these blocks before so might need some help to recreate them.

The process they are following is to create variables and these are covered in more detail in a future lesson.



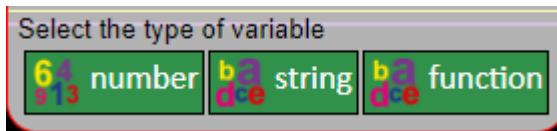
- 1) In the above example they firstly need to drag in the block . This block asks the user of the program a question - children type the question in. It then stores the result as



- 2) Then they should drag in the block




. 2Code will ask whether this should be a

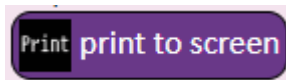


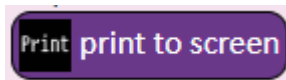
. For the example above select .

- 3) 2Code will name the variable myString1 because all variables must have a name. The name can be changed by deleting it and typing in a different name.




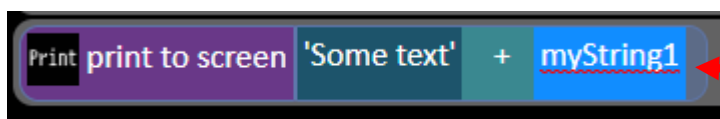
- 4) Click in the final box to see the variable to .



- 5) In some examples the  command is used to give the user information about the variables, sometimes this is combined with text. Use the options that pop-up to help you. Any variables that have been created will pop-up here, or you can type into the box. You can combine typing and variables by



using the  sign. To get these options to reappear, click on the end of the line of code:



Click here



# 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            | <p>Children have a basic understanding that coding involves writing instructions that a computer can follow.</p> <p>They are developing their understanding that these instructions must be precise and carefully structured through their work in Free Code Gibbon making simple one and two step programs for example in the lesson 1 bubble program or making an object move when clicked on in (lesson 1). Children know that an algorithm is related to giving instructions.</p> <p>With support, children can manipulate how their program looks using the 2Code design mode, by adding and changing backgrounds, characters, sounds (lesson 2) and objects. They can create a program that controls a character. They can make a character move when clicked but might not be able to plan how to make a character move when a different character (or the background) is clicked.</p> <p>Children are beginning to understand that they can correct unexpected outcomes by changing the code and they make attempts to identify the source of bugs.</p> <p>With support, children can explain the possible actions of objects including movement, clicking on them and collision. When looking at a simple program they can ‘read’ the code one line at a time but might not be able to envision the bigger picture of the overall effect of the program. Children will be able to suggest that an object might move when clicked but might not be able to suggest that an object might move when the background is clicked.</p> <p>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 3) – 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 4).</p> |
| Expected            | <p>Children can explain that an algorithm is a set of instructions to complete a task. They have turned algorithms of more than one step into code using freecode Gibbon. For example, in Lesson 1, they have been able to make a program that follows their algorithm e.g. ‘when the bubble is clicked it hides’. Children show an awareness of the need to be precise in their designs so that algorithms can be successfully translated into code.</p> <p>In lesson 4, children used a planning format on paper before implementing on screen within 2Code, they recognise this is the best approach for designing a solution efficiently. 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.</p>  |



## Assessment Guidance

|           |   |
|-----------|---|
|           | <p>They can use the Design Mode within 2Code to carefully see how their planned program will look and are able to switch into Code Mode to apply actions to objects. They confidently include objects, actions, events and outputs successfully within their 2Code programs.</p> <p>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 3). 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 3). Children can use ‘if’ statements to bring selection into their own coding (lesson 4). They understand how variables can be used to store information while a program is executing (lesson 6) and make attempts to use and manipulate the value of variables.</p> <p>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.</p> <p>Children can predict program outcomes and attempt to debug (lesson 5). Children can identify the parts of a program that respond to specific events and initiate specific actions. Based on this, children can predict and describe, using a cause and effect sentence, what will happen in a program. They can debug their own and other’s programs using design documentation to test against (Lesson 5).</p> |
| Exceeding | <p>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. Children can identify an error within a program that prevents it following the desired algorithm and then fix it (Lesson 5). Children make intuitive attempts to debug their own programs as they increase in complexity.</p> <p>Pupils realise the constraints of creating purely sequential programs and intuitively grasp the concepts of selection (lesson 4) and repetition (lesson 3). Children have a good understanding of when to use a timer in a program rather than a ‘repeat’ command to for repetition (lesson 3) 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 6).</p> <p>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. They exhibit greater ease at fixing their own bugs as their coding becomes more complex. (lessons 3,4,5 &amp; 6).</p>   |