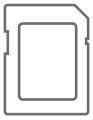
Scratch is visual programming environment. With it, you can create your own animations, games and interactive art works. And, while you're doing that, you'll learn some important principles and techniques of programming without actually having to write your own code. It's a great way to get started. To find out more about Scratch, visit the web address ***scratch.mit.edu***

Notes:

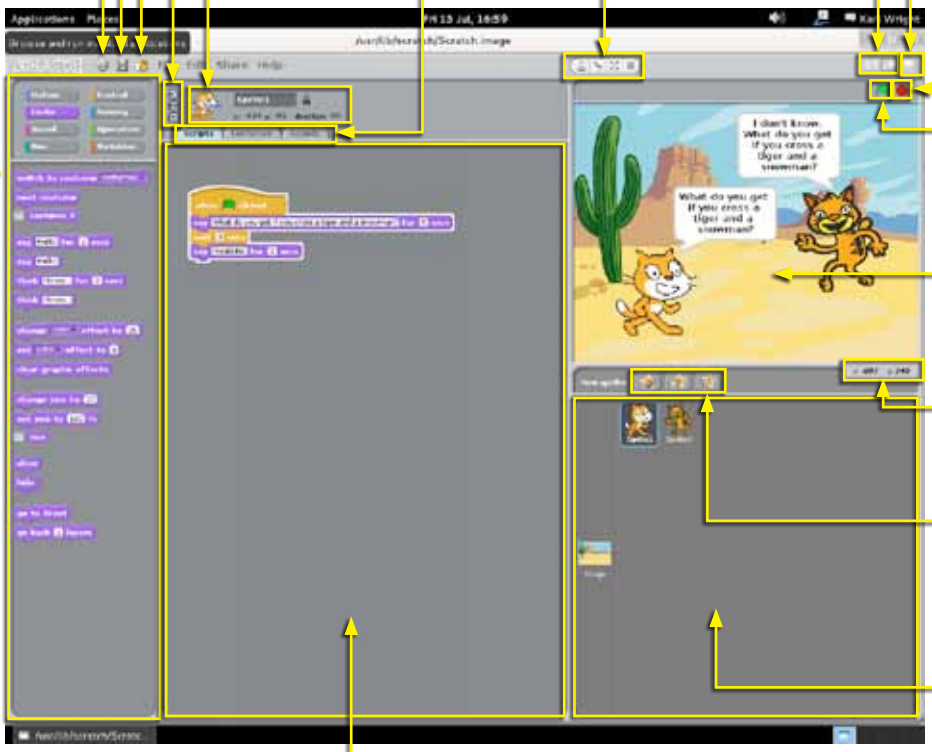
How to use this guide

We have tried to make this guide as straightforward to use as possible. To help you with the exercises in this chapter, we have already collected some little bits and pieces you will need, such as backgrounds, costumes for sprites, sound effects and complete examples of Scratch projects.



These can be found on the Raspberry Pi educational release SD card, in the folder /usr/share/scratch/RPiScratch. Wherever you see the SD card icon in the margin, that means we are referring to a file that can be found on your Raspberry Pi SD card. Go take a look! They can also be downloaded from Google Drive at ***http://goo.gl/MpHUv***

The Scratch interface



PRESENTATION MODE
Go fullscreen to show off your projects

SPRITE ROTATION STYLE

CURRENT SPRITE INFO

TOOLBAR

SHARE

SAVE

LANGUAGE

TABS
This is where you edit scripts, costumes or sounds

VIEW MODE
Change the size of the Stage

STOP SIGN
Stops your scripts

BLOCKS PALETTE
The blocks of code you'll use to program your sprites

GREEN FLAG
A way to start scripts

STAGE
Where your Scratch projects do their thing

MOUSE X-Y DISPLAY
Shows the location of the mouse cursor

NEW SPRITE BUTTONS
Create or import new sprites

SPRITE LIST
Find all your sprites here. Click one to select and work with it

SCRIPTS AREA
Drag blocks in, snap them together into scripts

Lesson 1.1: Scratch basics

Notes:

LEARNING OBJECTIVE: In this exercise, you will learn how to use the Scratch graphical user interface (GUI), how to create characters (sprites and costumes) and stages (backgrounds) for your projects, and how to add scripts.

RESOURCES: The sprites *"cat"* and *"roman_cat",* and the background *"roman_stage".*

Have you ever been in a school play? If you have, you'll know that to put on a play you need a stage, actors, costumes and a script. Think of Scratch as being a bit like a play. The actors are called "**sprites**".

You can dress your sprites in "**costumes**", and each sprite can have more than one costume. The "**stage**" is the area on the screen in which your sprites will perform the tasks you write for them.

To make your sprites move and talk, you need to give them instructions. You do this by writing "**scripts**" using blocks of code from the Blocks Palette and Scripts tab on the left of the screen.

That's enough introductions for now; let's get to grips with the program itself.

Open Scratch from your Raspberry Pi's Applications menu. You should now be looking at the Scratch graphical user interface, or GUI (pronounced "gooey"). Have a look around and tick the boxes below as you find these items:

- ☐ **1.** The stage (a big white screen)
- ☐ **2.** A sprite (clue: it's a cat)
- ☐ **3.** The two costumes that your sprite can wear (click on the Costumes tab)
- ☐ **4.** The Scripts tab

Click on the Scripts tab, can you see any instructions for the cat to follow?

Let's have some fun with the cat

First, let's give the cat something to say. We'll start with "Hello, World". This is generally the first thing a computer programmer learns to do (don't ask me why). As you are now learning a programming language, you'd better start with "Hello, World", too.

Notes:

Making the cat talk

To make the cat say "Hello, World", we're going to be working with "**blocks**". These are handy pieces of code, each containing an instruction for your sprite to follow.

There are eight different types of block. These can be found in the top-left corner of the Scratch GUI. They are colour-coded, so remember the colours. Find out what they are and complete their names in the table below:

M...	C...
L...	S...
S...	O...
P...	V...

Now, follow these simple steps to make your cat talk:

1 Click on the cat sprite in the Sprites List (bottom right) to make sure that it's selected.

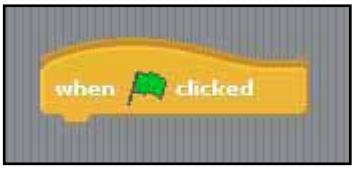
2 Click on the "Looks" button in the Blocks Palette to make the Looks blocks appear.

3 Click on the block labelled "say [Hello] for [2] seconds" and drag it to the Scripts tab.

4 Replace "Hello" with "Hello, World". Double-click the block and your cat should say: "Hello, World".

We have to run a program to make it work. You can do this by just double-clicking your script, if you only have one script. But if we have more than one script, we might want to start them at the same time. We can use a "green-flag event" for this.

Notes:



TO FIND THE BLOCK FOR GREEN-FLAG EVENTS:

1. Click on the Control button in the Blocks Palette.
2. Find the block labelled "when [picture of a green flag] clicked".
3. Select it, then drag and drop it to the top of the script you've created in the Scripts tab. Make sure it snaps into place.

You are now ready to run your first Scratch program properly. Just click on the green flag symbol at the top-right-hand side of the Scratch window, just above the stage, and watch the cat do its thing.

Over to you

QUESTION: For how long did the cat say "Hello, World"? _____ seconds
TASK: See if you can change the block to make the cat say "Hello, World" for 5 seconds.

Changing the way the sprite looks

Notes:

1 Click on your sprite to select it. In the Scripts area, click on the Costumes tab.

2 We are going to make a third costume for the cat, so click on Copy. A new cat costume should appear.

3 Select “costume3” and click on Edit. This will open the Paint Editor. Experiment with all the buttons and tools to find out what they do.

4 Once you feel at home, draw some clothes on the costume and click on OK. I gave my sprite a toga to make it look like a Roman Emperor.

5 Next, select the Scripts tab, click on the Looks button and select the “switch to costume [ ]” block.

6 Drag it under the Scripts tab and use the drop-down menu to select “costume3”. Double-click on this block and the cat will change his costume.

Now you have two blocks under the Scripts tab, one to say “Hello, World” and one for switching the costume. You can put them together by moving one so that it is just above or below the other. If a white line appears, the two blocks will snap together. Two or more blocks stuck together make a “script”.

Over to you

QUESTION: Now that we have a script with two blocks, what happens when you double-click it?

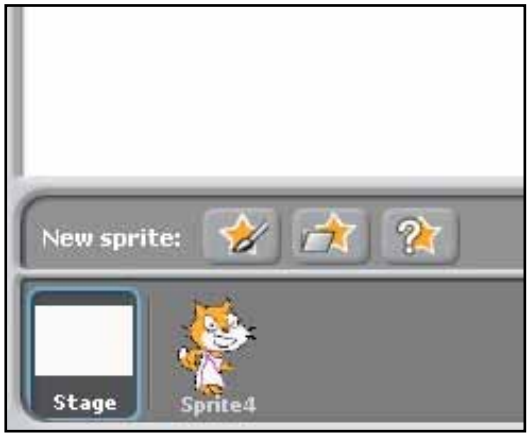
TASK: See if you can arrange three blocks to make the cat change to his toga costume, say “Hello, World”, then change back to its normal costume.

That cat’s right: he looks like he’s lost in a snow storm. We need to give him a stage on which to perform.

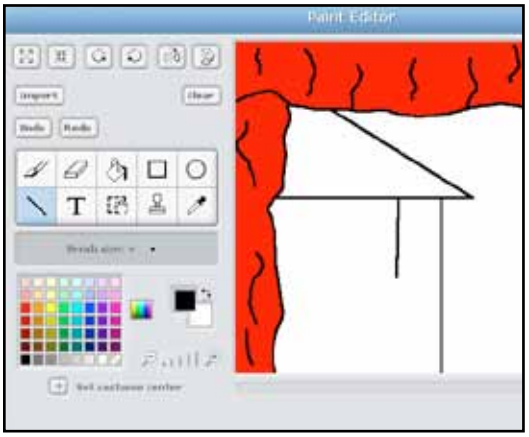
The stage

Notes:

It's time to give that cat a stage. We could be lazy and just import a picture to use as a background, but let's say that we're feeling energetic and want to draw our own.



1 Click on the stage in the Sprite List (bottom-right of the screen). Now click on the Backgrounds tab for the stage and click on the Edit button.



2 As before, the Paint Editor will open. Draw a stage for your sprite. When you have finished, click on OK. You can make further changes at any time by clicking on Edit.

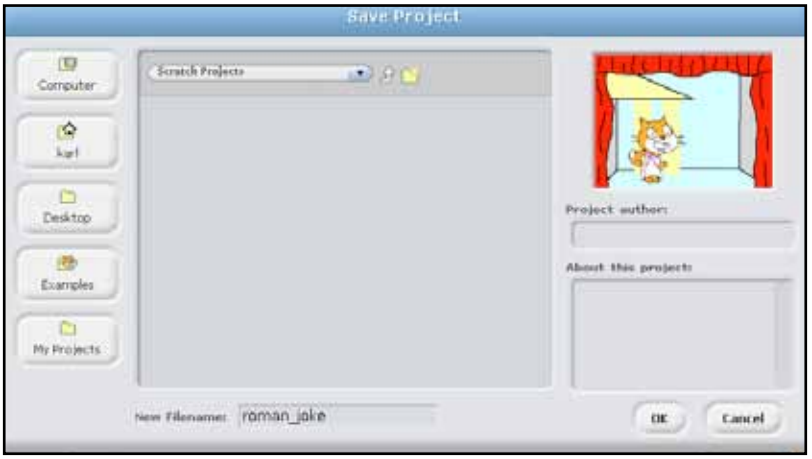
3 Alternatively, you can import a ready-made background. Select Stage, then Backgrounds and then click on the Import button.

4 Have a look at all the available backgrounds before you pick the one that you want. We chose "roman_stage". Select the background by clicking on it with your mouse, then click on OK.

Saving your work

This is a good time to save your project. You would be wise to do this every 10 minutes or so, then you can be sure that you won't lose any of your hard work. When working on a big project, save it in two places, then you have a backup.

To save your project, click File, then Save – the Save Project window will open.



By default, it will save your work to the Scratch Projects folder. This is a sensible place to store your work, so type in a new filename, at the bottom. I've called mine "roman_play", so pick a different file name for your project or you will save yours over mine! Click on OK to save.

Wow! That is a lot for the first lesson. Have a play with Scratch – experiment with different blocks of code to find out what they do. Then come back when you have had a good rest and try Lesson 2.

Notes:

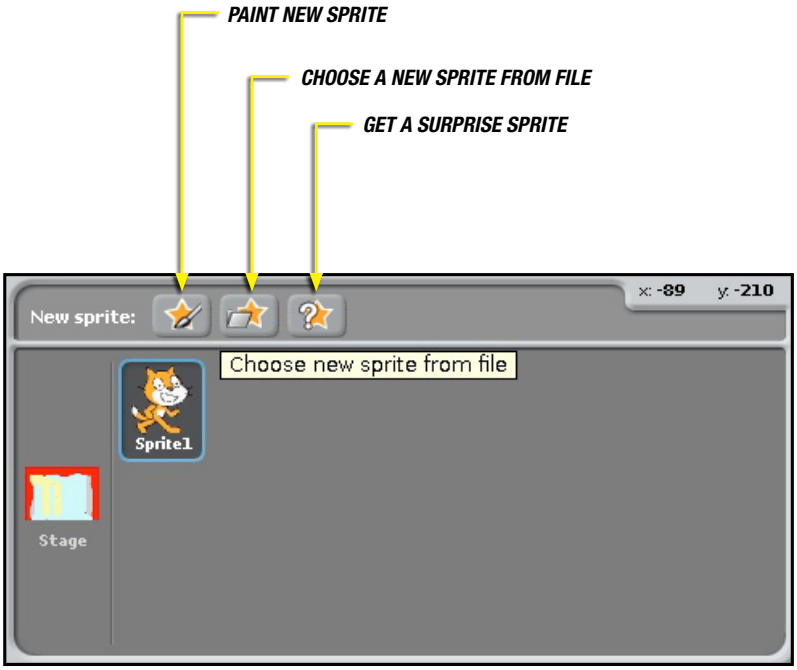
Tip...

Use a name that will help you to find the project again. Always use an _ (underscore) between words in filenames – don't leave an empty space.

Notes:

LEARNING OBJECTIVE: In this exercise, you will learn how to move sprites around the Scratch screen in a controlled way and how to tell a joke.
RESOURCES: The sprite "*roman_cat*" and the background "*roman_stage*".

The cat is feeling a bit lonely, so we'd better create some characters for it to play with. You can either paint your own sprites or import sprites from the Scratch Costumes folder. Use the New Sprites buttons to do this.



On the right-hand side of the program, just below the stage and above the Sprite List, you'll see three buttons: the New Sprite buttons. It's these we're going to use.

I want to add a time-travelling boy to my stage. To keep things simple, and to let us get on with some more programming, we're just going to import him.

Click on the middle New Sprite button and import the sprite "*boy4-walking-c*". But wait a sec: he's facing the wrong way! No problem. Go to the Costumes tab and click on Edit. Use the Flip Horizontally button to make him face to the left.

Use the Import Sprite button to find and import the sprite "boy4-walking-c".

From the Costumes tab, click Edit and use the Flip Horizontally tool.

There are also buttons to make your sprite bigger, smaller, rotate counter-clockwise, rotate clockwise, as well as flip horizontally and flip vertically. Try them out. I have also used the shrink button to make my boy smaller.

Make your sprites tell a joke

Let's make the sprites tell each other a joke. You can do this using the speech block from the Looks category.

You could try a simple 'knock knock' joke to start with.

But wait! Are you finding that both of your sprites are talking at the same time.

To fix this, from the Control block add the "wait [1] secs" block to the second sprite, before the "say" block.

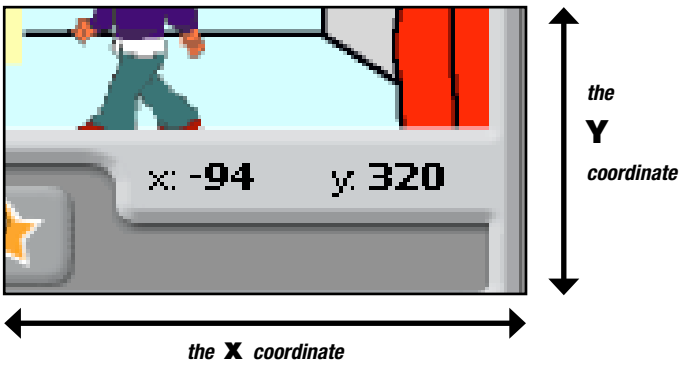
Notes:

Positioning your sprite

Notes:

Ok, we've told a joke. But this play is looking a bit static, so let's make our characters move. The first job is to move our two characters to their start points. In my play, the cat will come in from the left and the boy from the right.

You may have come across x and y axes when creating graphs.



The coordinates of any point on the stage are shown at its bottom-right-hand corner. Move your mouse around the screen and watch the numbers change.

TASK: Use your mouse to find the centre of the screen. Move the mouse pointer until it's exactly over the point x: 0 y: 0. Now let's position our sprites.

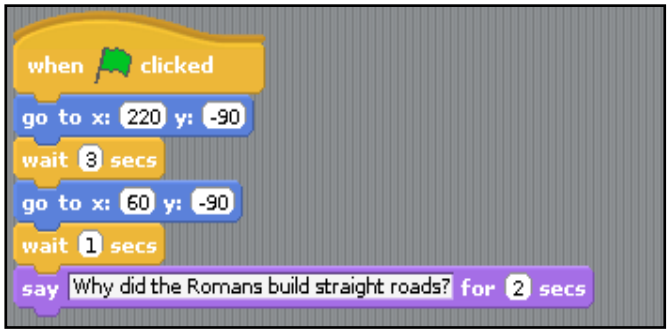
1. Select the cat sprite then, in the Blocks Palette, click on the Motion block labelled "go to x: [0] y: [0]".
2. Change the values in the block to x: -240 y: -80. This will take the cat to the far left of the stage.
3. Next place a "wait [1] secs" block into your script. This will give you time to see your cat before it moves.
4. Now add a second "go to x: [0] y: [0]" block. Use your mouse to work out the x coordinate just left of centre on the stage, to which we want to move the cat.

Repeat this process for your other sprite, positioning it slightly to the right of centre stage. Ideally, the two sprites should move from the edges of the screen to stand face to face, separated by a small gap.

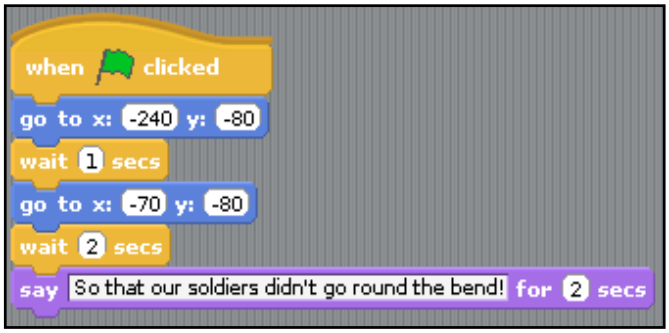
Now you need to make the sprites tell a joke. Remember to leave a short delay after each sprite speaks, otherwise they'll talk over each other. Have a look at the screenshots to see our code (and our fantastic joke).

Tip...

Use your mouse pointer to find the coordinates of a position on the stage and make a note of those coordinates on a piece of paper.



This is what our code for the boy sprite looks like.



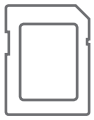
And here's our code for the cat sprite. Does your looks the same?

Over to you

Notes:

TASK: Now add some code to your other character to move it to the right of the stage and then after a short delay move it into the centre stage.

Well done! You have certainly got the hang of moving sprites about the screen. Why not add some more characters to your stage and get them to tell jokes?



*If you are having problems, you can load the sample code, "*roman_play.sb*", to see how the program is put together. Feel free to change things and to experiment, as this is a great way to learn.*

Lesson 1.3: Animation (loops)

LEARNING OBJECTIVE: In this exercise, you will learn how to use repeat loops to create simple animations.
RESOURCES: The sprites "*bee*", "*female_flower*" and "*male_flower*", and the background "*flower_bed*".

With its animated characters, Scratch is great for telling stories. I have to do a school science project on pollination, so I have decided to use Scratch to tell the story of pollination in moving pictures. You can help me by following these instructions to animate a bee in flight.

First, open the file "*bee1*" from the "*Animals*" folder in the Scratch gallery. Next, import the background "*flower_bed*", this time from the "*Nature*" folder in the Scratch gallery. Delete the cat sprite; we don't need it for this project.

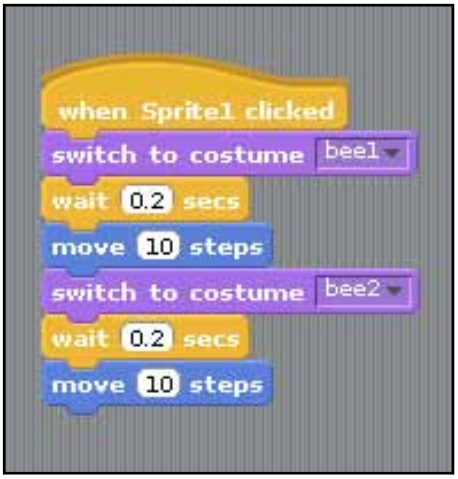
Copy "*bee1*", then edit "*bee2*" using the Select and Flip Horizontally tools, to make its wings point downwards. Together, the two costumes – "*bee1*" and "*bee2*" – will become an animation of a flying bee.

Copy your bee, then edit "bee2" so that its wings point downwards.

Notes:

We need some script to make the bee look as if it is flying. We do this by switching from one costume to another and back again, making the bee appear to flap its wings. As we do this we will also make the bee move forwards.

Now, build your own script to make the bee fly. You will need blocks from Control, Looks and Motion. If you get stuck, have a look at the screenshot of our code. You'll find it further on in the lesson.



This is the code to make your bee fly. Instead of using the green flag to run my code, I will use a "when Sprite1 clicked" block from Control. The code will run when I click on the bee.

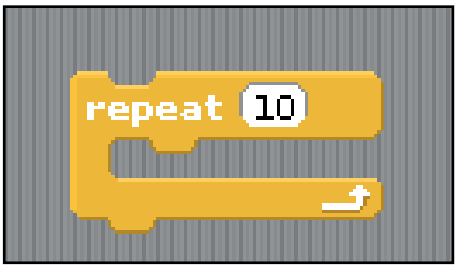
Here are the steps you need to follow:

1. Start with costume "*bee1*".
2. Add a "wait [0.2] secs" block, so that the viewer has time to see the costume.
3. Move the bee on 10 steps, before switching to costume "*bee2*".
4. Add another "wait [0.2] secs" block, so that the viewer has time to see the second costume.
5. Move the bee on another 10 steps.

But we need to do this more than once. To make the bee fly across the screen, we might have to repeat this 20 times.

Don't panic! You are using a computer. Computers are fantastic at doing things over and over again. They can do this very accurately and never get bored, tired or fed up.

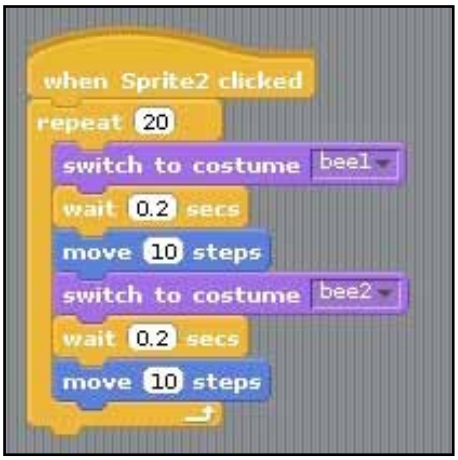
What we need is a repeat loop.



This is what we use to program the computer to repeat something over and over again. You will find the repeat loop ("repeat [10]") in the Control blocks.

It looks a bit different because it has a gap so that we can put code you want to repeat inside it. Just drag the "repeat [10]" block to sit directly under the "when Sprite1 clicked" block. It will automatically fit around your block of code, causing it to repeat itself.

Here is the code we need to animate the bee so that it flies all the way from one side of the screen to the other.



Using the repeat loop, you can make the sprite do the same actions over and over again. So the bee flaps its wings up and down many times.

Over to you

QUESTION: Why do you think I have increased the number of times it repeats from 10 to 20?

TASK: Some of the Scratch sprites already have two costumes. Check out the Scratch cat sprite. Use its two costumes and code similar to the example on the left to make it walk.

Notes:

Tip...

If your bee flies in the wrong direction, check the sprite to make sure that it is facing in the correct direction. With the sprite selected, look at the bar above the Scripts tab. The "forward" direction of a sprite is indicated by a little blue line. The bee on the left will move 90° to the vertical and the bee on the right will move 60° to the vertical. You can rotate the line to change the move direction of a sprite.

Notes:

This is the storyboard of my pollination project. I added two more costumes to my bee sprite to show it carrying pollen and I have drawn a flower sprite with some stamens in blue.

But I didn't want to stop there. I wanted the bee to visit the second flower – a female – from the other side of the screen, so I copied all four costumes for the bee sprite and flipped them horizontally.

I also copied, flipped and edited the male flower to create a female flower sprite. I have given it two costumes. One shows the stigma without pollen and the other with pollen. Let's have a look at the resulting animation.

1. *A bee flies towards a male flower.*

2. *The bee pauses to suck up nectar and collect pollen.*

3. *The bee then flies off with pollen from the flower's stamens.*

4. *The bee flies toward the female flower.*

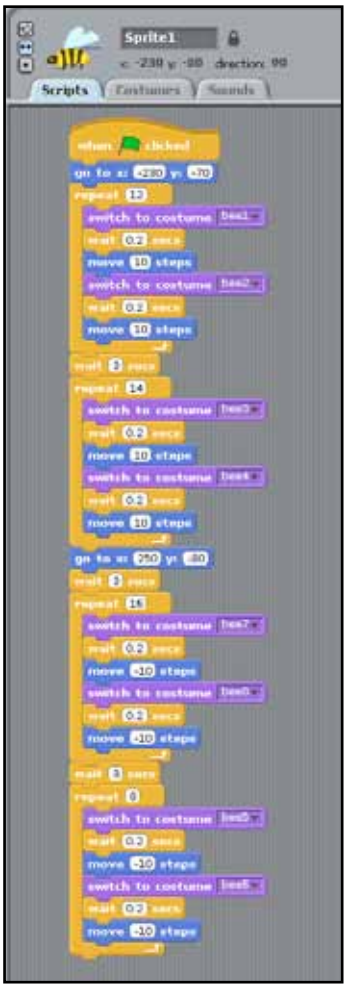
5. *The bee sucks nectar and this time deposits pollen.*

6. *The bee flies off leaving pollen on the stigma.*

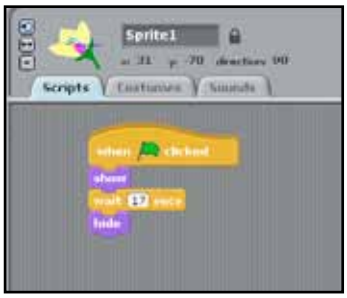
I also decided that I only wanted one flower on screen at a time. So, I had to add scripts to make my flowers disappear and appear at the right points in the animation. I used the Looks blocks "show" and "hide" for this.

Here's the code for all three sprites in the pollination project.

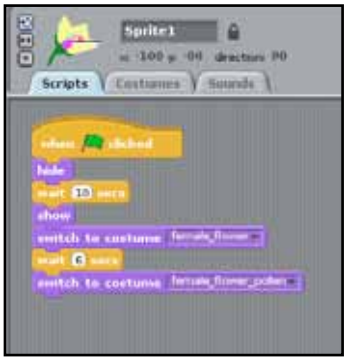
Code for the bee



Code for the male flower

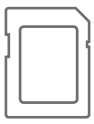


Code for the female flower



Because I had three different sprites with their own scripts, I used the "when [green flag] clicked" event to run them all together.

Wow! That is quite a complicated project, but if you break up your animation into lots of little scenes it makes it easier to plan and to program.



To see what the whole project looks like once it's finished, open RPiScratch/Projects/ pollination.

Notes:

Import existing sprites and a background from the Scratch picture folders. This can save you a lot of time.

Notes:

LEARNING OBJECTIVE: You will learn how to use variables to store data for using in a program. You will also learn how to use operators to do simple sums.
RESOURCES: The default sprite "*cat*" on the default white background.

Do you find maths difficult?

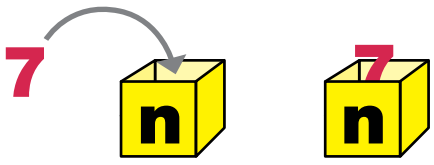
Can you imagine what it would be like to be able to do millions of sums in seconds and always get them right? Even the most complicated sums you can think of?

Computers are fantastic at maths. In fact, maths is what they do best. We can program the Scratch cat to do maths. The cat will ask for some numbers and then do the sums. So, how are we going to put numbers into the program for the cat to use?

When we input numbers (put numbers into a computer), the computer has to have somewhere to store them. Different people might input different numbers, so these numbers are going to be different each time.

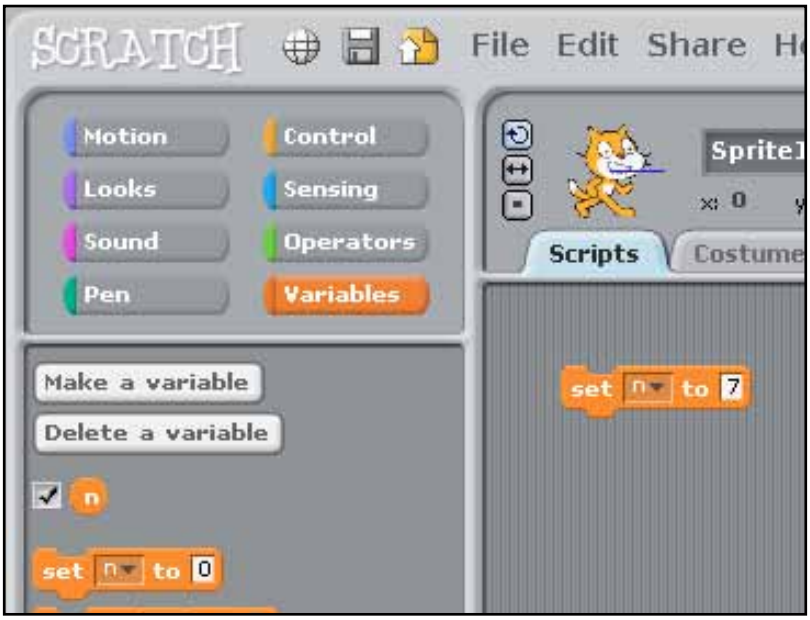
When programming, we store numbers in something called a "**variable**". One way of thinking about a variable is as a box, or container, in which we can store numbers, letters or words.

We may have more than one variable in a program, so we give them different names. The name can be as simple as a single letter (or as complicated as you like!). For example, if it is storing a number, we might call the variable "n".



In the diagram, we have stored the number "7" in the variable "n". So we can now say "n = 7".

In the example above, we created a variable called "n" and stored the number "7" in it. In Scratch, you would do this in two steps: first creating the variable "n", and then using a block from Variables to set its value to "7".



If you wanted to use a two-word name for your variable, you would separate the words with an underscore (the "_" character), not a blank space.

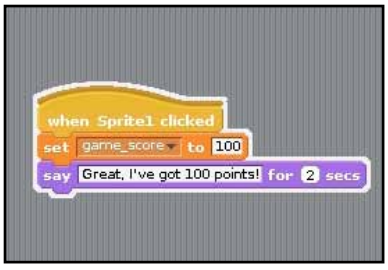
Using variables in Scratch

Ok, now we're going to create and use variables. Click on Variables in the Blocks Palette and create a variable called "game_score".

1 Click on the Variables button, then on the button labelled "Make a variable". This opens the "Variable name?" dialogue window. Enter the name "game_score" for your variable and click OK.

2 Drag "Set [game_score] to [0]" to the Scripts tab. Then, from Control, drag the block "when Sprite1 clicked". Join the two together to make a script.

3 The default value for new variables is "0". Select the block "set [game_score] to [0]" and change the value to "100".



4 Finally, from Looks, I have used a "say [Hello] for [2] secs" block but changed it to "say [Great, I've got 100 points] for [2] secs".

Notes:

Tip...

Give each variable a name that reminds you what is stored in it. For example, if you are creating a game and you use a variable to store the score, then a good name for the variable would be "game_score".

In this example, we set the cat's score to 100 points by clicking on the cat sprite. But there are plenty of other ways to put a number into a variable. The method you choose will depend on how you want to use the variable.

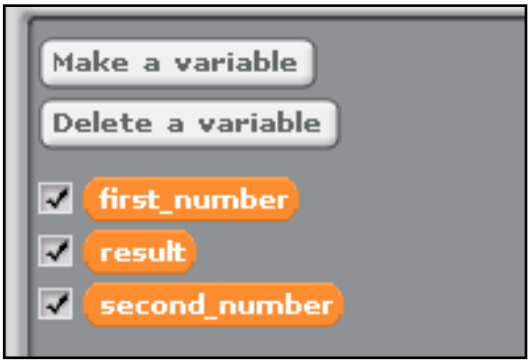
Notes:

Inputting the numbers

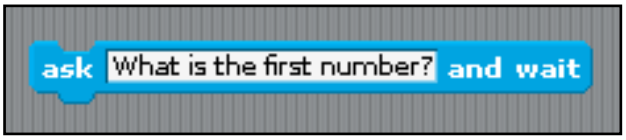
Your teacher has set you four sums. The sums are:

56 + 39 =
87 - 42 =
16 x 9 =
240 ÷ 6 =

We're going to show you how to get Maths Cat to do this homework for you. Let's start with the first question, 56 + 39. If you get stuck, refer to the image of the completed code at the end of this exercise.



First, create three variables, and name the first two "first_number" and "second_number". You'll use these to store your two numbers. Name the third variable "result". This is where you'll store the answer to your sums.



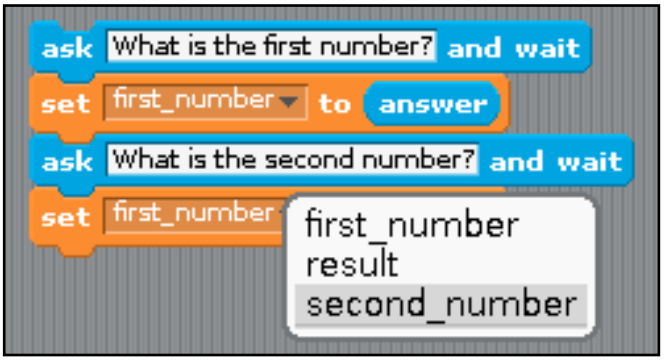
To input the first number you need to use a Sensing block to tell the user what to do. Drag the block "ask [What's your name?] and wait" to the Scripts tab. Change the value to "What's the first number?".

The answer given by the user is then entered to a variable, where it's stored for use in your sum, by using a combination of a set "variable" block from Variables and an "answer" block from Sensing. Let's see how.

Click on Variables, select the block "set [first_number] to [0]", drag it to the Scripts tab and snap it to the previous block.

Notes:

Click on Sensing, select the block labelled “answer” and drag it onto the number ”0” in the previous block. Your script should look like the one in the screenshot above.



Now the program knows the first number in your sum. To tell it the second number, repeat the process above but remember to change “first_number” to “second_number” in the drop-down box of the variable block.

Now for the really clever bit

To do sums in Scratch, you need something called an “**operator**”. That’s just a fancy term for maths signs such as +, -, × and ÷. Yes, you guessed, we find these under the green Operators category in the Blocks Palette.

Click Operators: the “add” operator (“[ ] + [ ]”) is the first one on the list in the Blocks Palette.

We’re going to use the “add” operator, so drag it to the Scripts tab. We’ll use it to add the variables “first_number” and “second_number”.

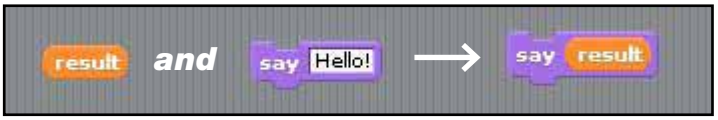
From Variables, grab the variable “first_number” and drop it into the first blank space in the “add” operator. Drop “second_number” into the second blank space (see the screenshot on the next page).

We're not finished yet. We've told the program to add our numbers, but we need to store that answer in one of our variables.

From Variables in the Blocks Palette, drag "set [first_number] to [0]" to the Scripts tab. Change "first_number" to "result" and drag your "add" operator onto the number "0". Your block should look like this:

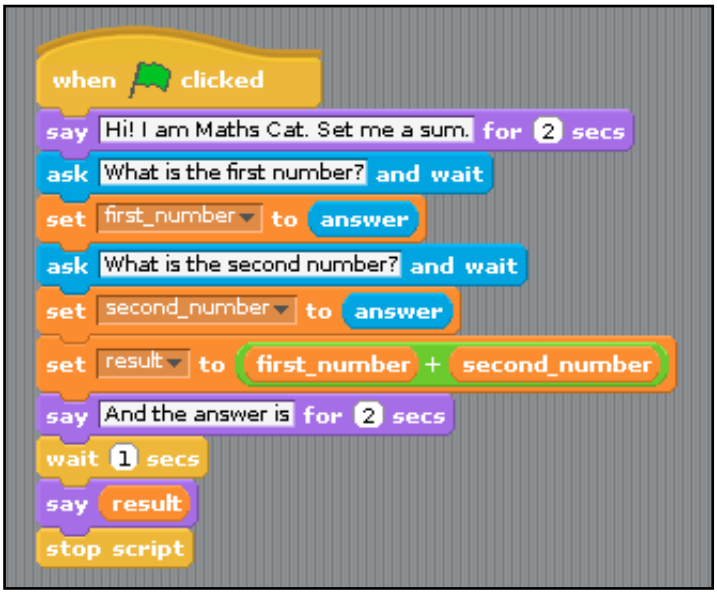
Finally, we also need to tell Scratch to display the answer to the sum. Otherwise, it will keep it to itself (and that's no good to us).

Click on Looks in the Blocks Palette. Select "say [Hello!]" and drag it on to the Scripts tab. From Variables, drag the variable "result" and drop it onto "Hello!". The image below will show you what we mean.



And that's it! I have included the whole program below. We've added in some extra blocks, to make the cat a bit chattier but the basics are the same as the script we built above.

Use this screen-shot to build the whole script. And remember, each time you want the cat to do another sum you will have to click on the green flag.



But hang on a minute: if you tried to make the cat do all the homework you will have noticed a big problem.

The cat can only do addition! Don't panic – a quick edit to change the program slightly will do the trick.

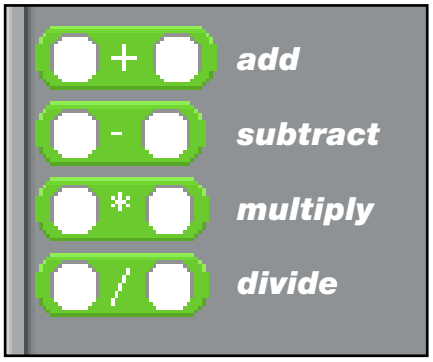
Tip...

Make sure all the blocks snap into place properly, especially when you have to snap blocks on top of other blocks.

Notes:

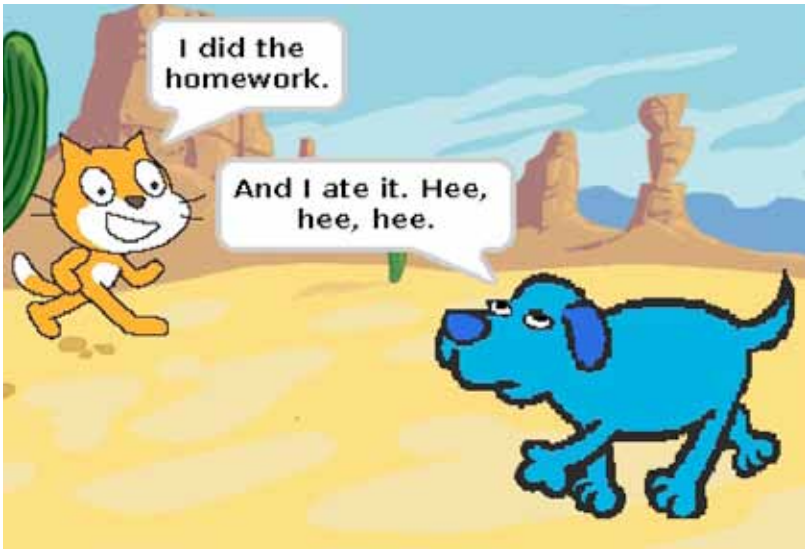
Notes:

Click on Operators once more in the Script block and you'll see that there are other operator blocks there too: for subtraction, multiplication and division.



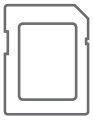
If you want the cat to take away then simply swap the "add" block in the script above for a "subtract" block, and so on until the cat has done all you homework for you.

What a helpful cat!



Over to you

TASK: Program the Maths Cat to do the rest of the homework sheet – you will have to change the operator each time so that the cat does the right kind of sum.



To see a working example of this script, open RPiScratch/Projects/maths_cat.

Lesson 1.5: Artificial intelligence

Notes:

LEARNING OBJECTIVE: In this exercise, you will learn how to put data into a program and get your program to make a decision based on that data.
RESOURCES: The sprites "*cat*" and "*bluedog*"; the backgrounds "*brick-wall1*", "*sydney*", "*paris*" and "*new_york*".

Artificial intelligence is an area of computer science where people try to make computer programs that are smart in some way or another. The idea is to make computers seem like they are thinking like humans. This is actually quite tricky, as you can imagine, so here we are just going to give you a tiny taster of how you can make your programs seem a bit intelligent.

The cat clearly thinks it's clever, so let's give it a chance to show us just how intelligent it is. To do this, we will use some more inputs and outputs, together with something called a "conditional statement". That sounds very complicated, but it isn't really.

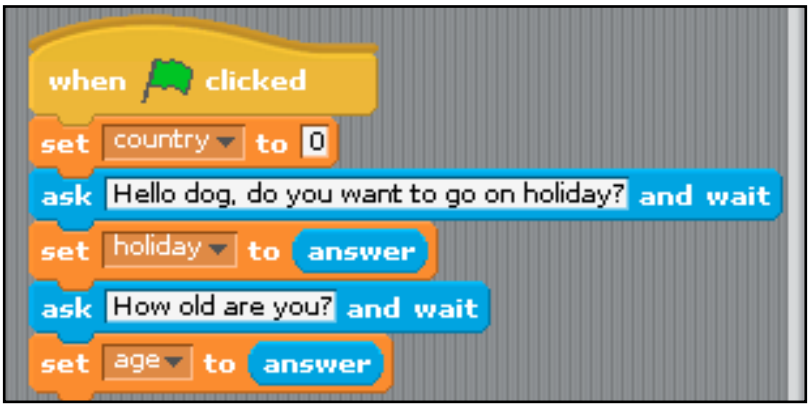
For my example, I have created two sprites: a cat and a dog. The cat is going to ask the dog a number of questions, so we need some variables in which to store the answers.

Create the following variables (we'll tell you what they're for in a minute):

age
country
holiday
name

Before we begin, you should also import the "*bluedog*" sprite and the backgrounds "*brick-wall1*", "*sydney*", "*paris*" and "*new_york*". You will need them for what comes next.

Does the dog want a holiday?



First, we have to work out whether the dog actually wants to go on holiday. Select the cat sprite and build the script you can see in the screenshot above to help the cat find this out.

Each time we use this program the country variable will be set to a letter. But the next time we use the program, we want the country variable to be empty. So we "empty" it by setting it to "0", ready for the next user.

Notes:

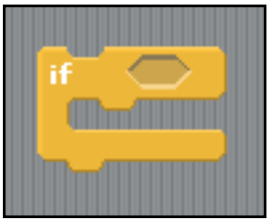
QUESTION: Study the script. Can you work out what it will do when the program is run?

As the cat asks questions, the dog's answers are stored in the two variables "holiday" and "age". The script will use these later.

Now for the intelligent part: the conditional statement. We are going to find out if the dog wants to go on holiday. Time to think logically: the dog will either answer "yes" or "no".

We need to find a way of letting the cat know the dog's answer and of prompting the cat to act on that answer.

From Control in the Blocks Palette, select an "if" block.



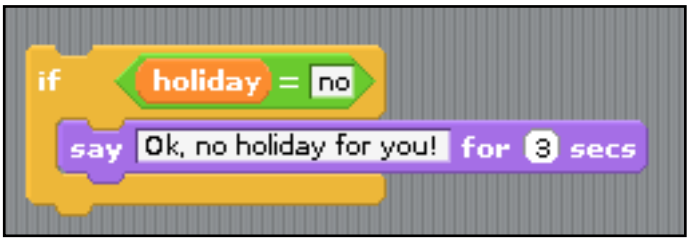
You will also need an operator block from Operators to test the input. Grab the "equals" operator ("[ ] = [ ]") from the Blocks Palette.

From Variables in the Blocks Palette, drag and drop the "holiday" variable into the left-hand side of the operator and type "no" into the other side.

Finally, we drag and drop the operator block onto the "if" block.

Now, from Looks place the block "say [Hello] for [2] secs" inside your "if" block – that is, in the "bracket" cut into the side of the block, so that the "if" block surrounds it. Change "Hello" to "OK, no holiday for you!".

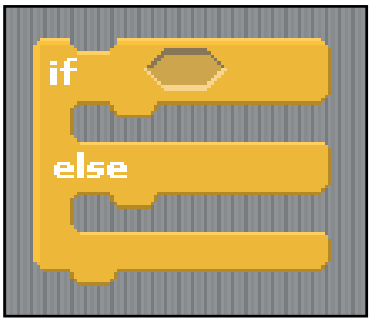
The code to find out if dog wants to go on holiday



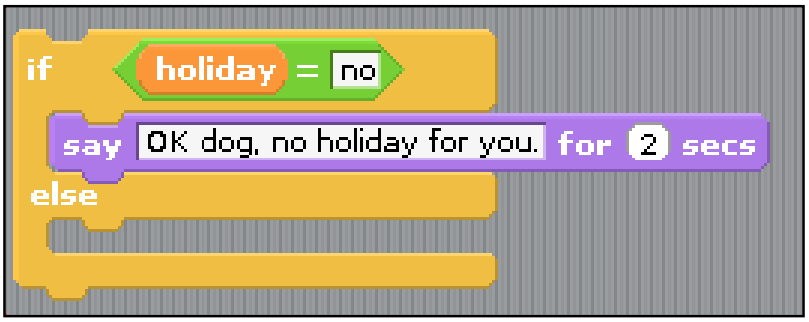
But hang on a second, what if the dog does want to go on holiday? Our "if" block doesn't allow us to include a possibility for a "yes" answer.

If the dog does want to go on holiday something else must happen. We need to replace our "if" block with an "if/else" block.

The "if/else" block.



Take the little script that was in your "if" block and put it into the "if" bracket of an "if/else" block. The result should look like the screenshot below.



Is the dog old enough to go on holiday?

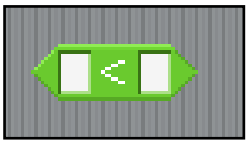
We also need to find out if the dog is old enough to go on holiday by herself. We will need to use a second "if/else" block. Drag it onto the Scripts tab, but don't attach it to your script just yet – leave it floating by itself.

We're going to ask the dog how old she is. Depending on her answer, there are two possible outcomes:

1. Her age is less than 10, and she's not allowed to go on holiday.
2. Her age is greater than 10, and she is allowed to go on holiday.

If the dog's age is less than 10, we want the cat to say, "Sorry dog, you're too young to go on holiday."

The "less than" operator.

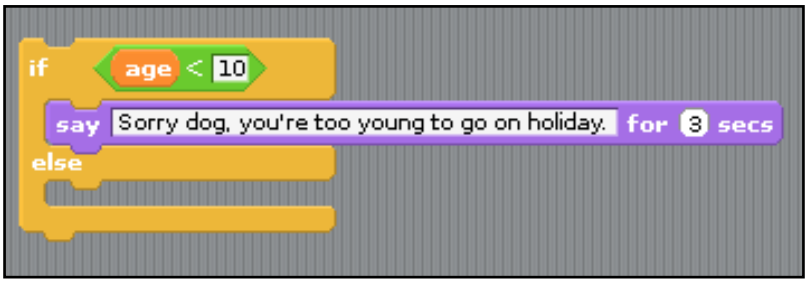


Using your "less than" (<) operator, as well as blocks from Variables and Looks, build the piece of script that you see in the screenshot on the next page.

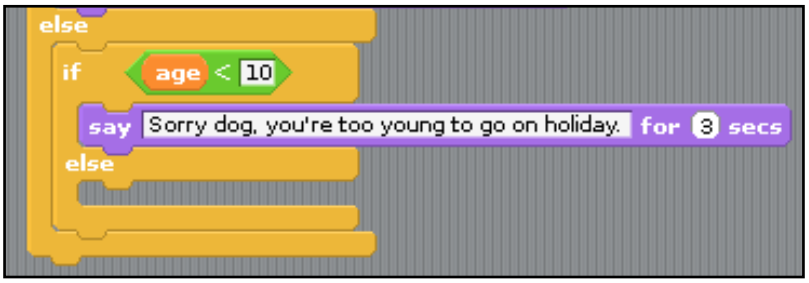
Build this script inside the "if" bracket of your second "if/else" block.

Notes:

Notes:



Before we move on to the next step, take your second “if/else” block and place it inside the “else” bracket of your first “if/else” block: the one you used to find out if the dog wanted to go on holiday.



Your script should now look like the block of code in the screenshot above.

Over to you

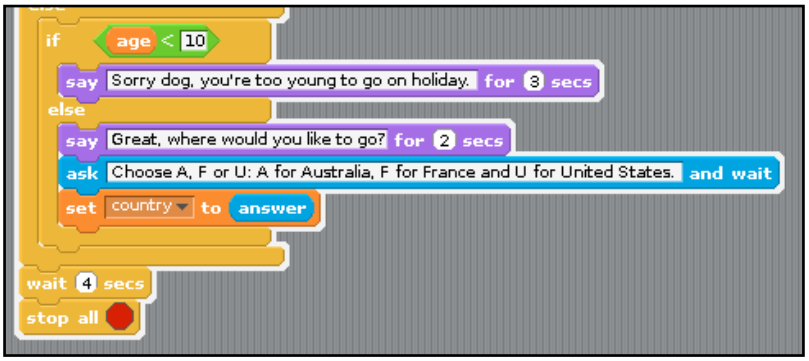
QUESTION: What will the cat say if the dog’s age is 9?
QUESTION: What will the cat say if the dog’s age is 12?
QUESTION: What will the cat say if the dog’s age is 10?

Where does the dog want to go?

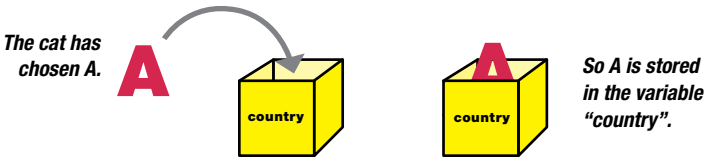
To be fair to the dog, we should let her choose where to go on holiday. We will give the dog three possible holiday destinations: Australia, France and the USA. To keep it simple, we won’t ask the dog to enter the country’s name, just the initial letter.

Using the information you can see in the screenshot below, add blocks to your script to make it ask the dog where she wants to go on holiday. Store the dog’s answer in the variable “country”.

You will need blocks from Looks, Sensing and Variables. Place them in the “else” bracket of your second “if/else” block.

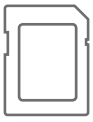


Finally, complete your script with a “wait [ ] secs” block and “stop all” block from Control. Set the value of the “wait [ ] secs” to “4” and place both blocks at the very end of your script, outside both “if/else” blocks.

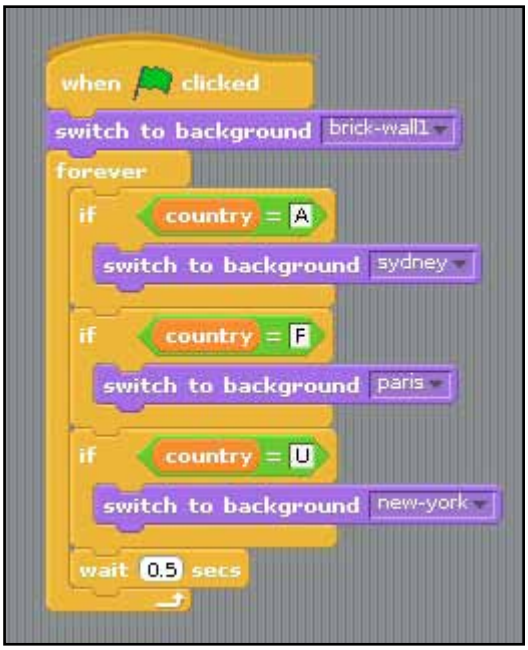


The destination the dog chooses will be stored in the variable “country”. This is important, because this variable will be used by the script for the stage to determine the script’s output.

The script for the stage



To complete the program we need to program the backgrounds. If you haven’t imported them from RPiScratch\Resources\Backgrounds, now’s the time to do it.



Select the stage in the Sprites List and build the script you can see in the screenshot above.

Here, we have created three “nested if” conditional statements, which will switch the background according to the dog’s choice. We use the term “**nested**” when one conditional statement is put inside another one.

Each time we use this program, the stage has to be reset to the “*brick-wall1*” background. The conditional statements need to keep checking until the dog has made her choice, so I have put them in a “forever loop”.

Notes:

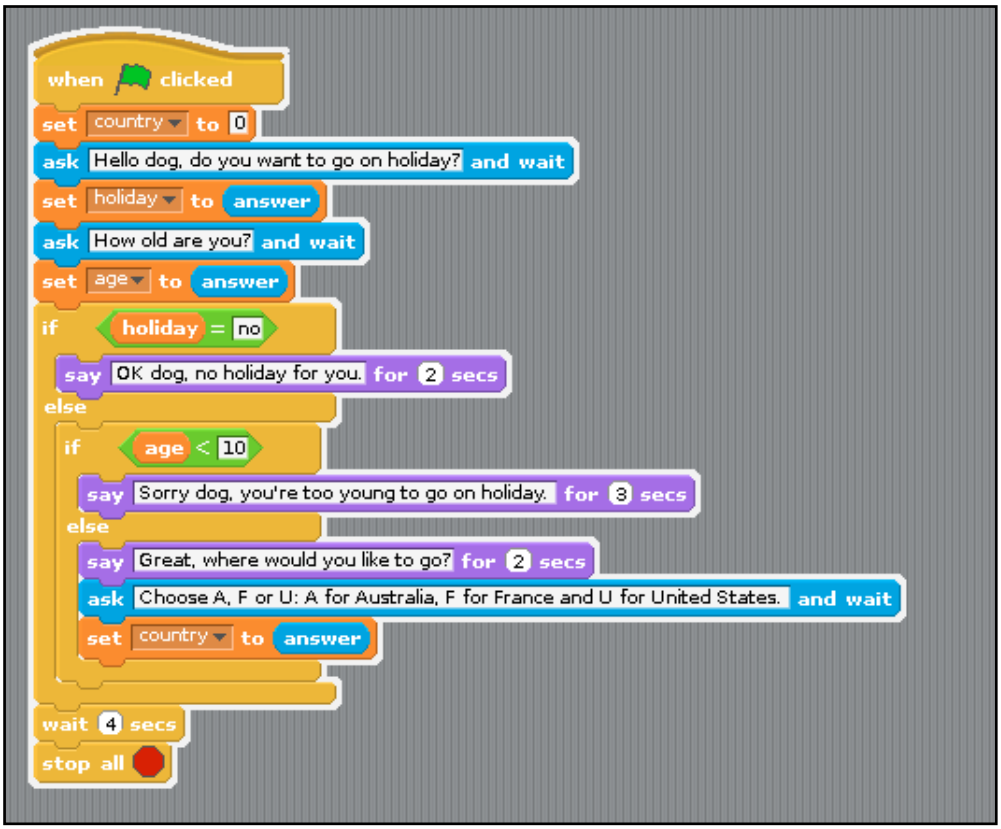
Tip...

Users do not always do what they are supposed to do. If a user provides an unexpected input, this can cause an “error” in the program, which will often cause the program to “crash”. Programs that have errors in them are said to contain “bugs”. An important task for a programmer is testing their program to make sure it is bug-free.

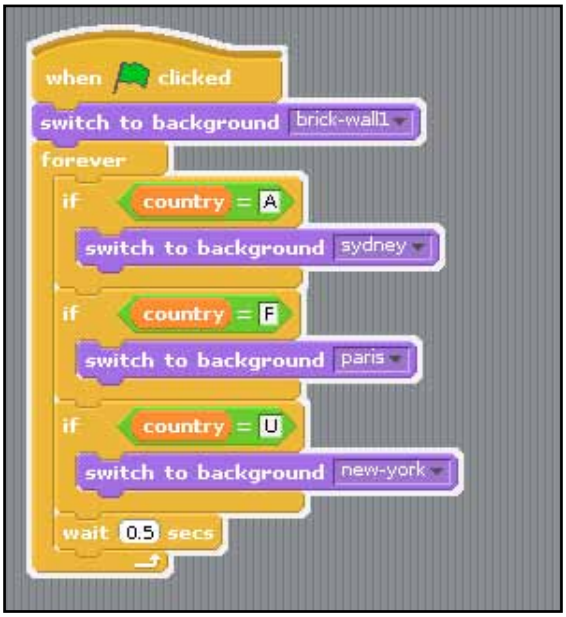
The full script

Phew, that was complicated. But hopefully you managed it all in the end. Just to make things a bit easier, here are the full scripts for both the cat sprite and the stage.

The code for the cat



The code for the stage



To see a sample of the code for this lesson, open RPiScratch/Projects/holiday_dog.

Notes:

Over to you

Notes:

QUESTION: Study the code shown above for the cat sprite and the stage. Do you understand it?

TASK: Experiment with the code. See if you can add some extra questions.

A final word

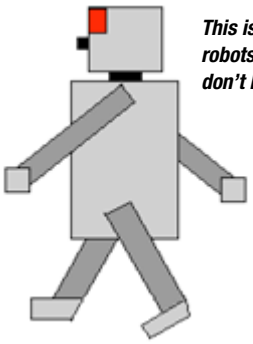
In fact, the cat is not intelligent at all. Computers have no intelligence – it is the programs that run on them that make them appear intelligent. This is why we use the term "artificial intelligence". The only intelligent things here are the programmer, who programmed the cat, and you, for completing Lesson 1.5.

Lesson 1.6: Control

LEARNING OBJECTIVE: In this exercise, you will learn how to write control programs that respond in different ways depending on inputs to the program.

RESOURCES: The sprite "*robot_up*" and the background "*green_background*".

The robot



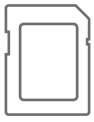
This is how we think of robots, but most robots don't look like this.

You're probably used to robots from the movies: metal men clanking and talking in metallic voices. This is actually an old-fashioned idea of a robot. It dates from a play written in 1920. I bet even your teacher was young then.

A more modern way of thinking about a robot is as anything that can be controlled by a computer. This is known as "control engineering". This device could be an aeroplane, a washing machine, a lathe, a welding machine, a level-crossing barrier, a sewing machine, a self-drive car or anything else you can imagine.

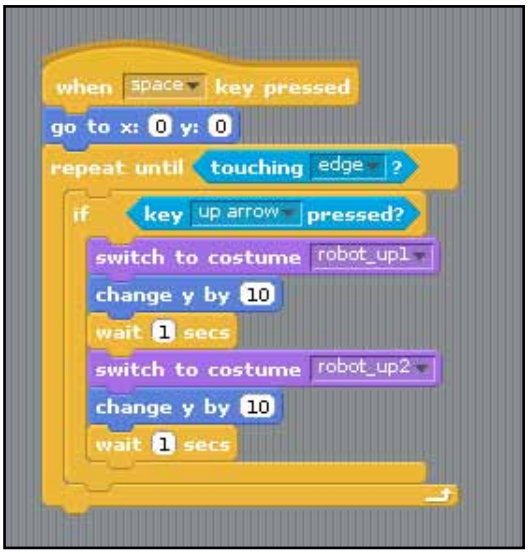
But let's start with a proper, old-fashioned robot. We don't have a real robot, or all the wires and circuits we'd need to control one. So, instead, we'll use a robot that lives inside the Scratch stage. What you learn by doing this is how to make your robot respond to inputs.

We are going to program our robot so that we can control it using the "up arrow" key. To make it easier to see our robot moving, we need to be able to see him from above.



Import the sprite "*robot_up*" from RPiScratch/Resources/Costumes. While we're at it, import the background "*green_background*" from RPiScratch/Resources/Backgrounds.

This script will make our robot move on our command, but only in one direction.



Have a look at the screenshot above. We have used the "when [space] key pressed" event from Control to run the program. The robot sprite is moved to the centre of the screen using a "go to x: [0] y: [0]" block from Motion (see "Positioning your sprite", in Lesson 1.3).

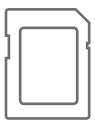
The rest of the code is placed inside a "repeat until" loop (from Control). Inside the repeat loop, we have an "if" (a conditional statement) that checks if the "up arrow" key has been pressed. This has been set to a "key [up arrow] pressed" event (from Sensing).

The "touching [ ]?" block from Sensing is set to "edge". That means that if the robot touches the edge of the Scratch screen, the event will stop the script. An "event" is something that happens in a program.

In this script, there are two possible situations:

1. The "up arrow" key has been pressed – in which case, the robot will walk up the screen for 20 steps, switching costumes as it does so.
2. The "up arrow" key has not been pressed – in which case, the robot will do nothing.

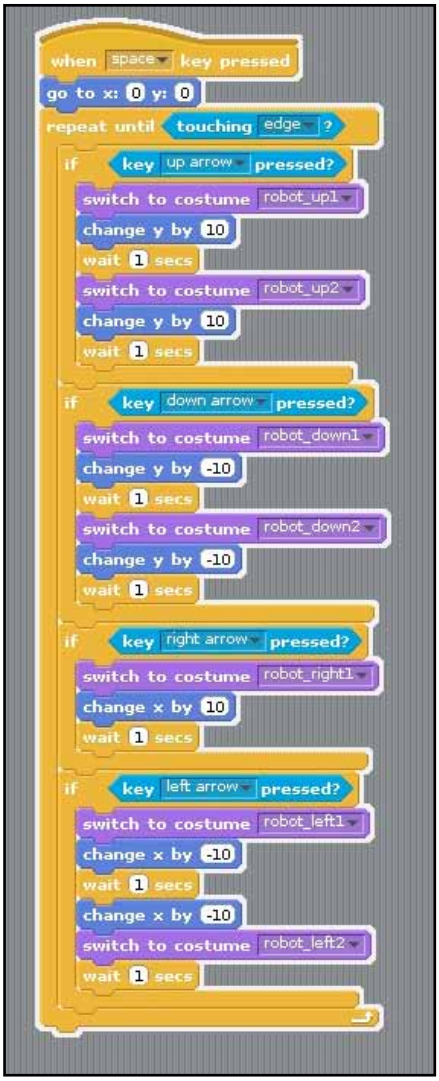
That's not going to be very useful. Let's see if we can do better.



To see an example of this script, open RPiScratch/Projects/robot_v1.

Notes:

Notes:

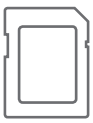


RESOURCES: The sprite "*robot_control*" and the background "*robot_maze*".

We have included the code of a more complete version of the robot program, "Robot_2" (see the screenshot above). In this version, the robot can be moved up, down, left and right.

TASK: Now it's your turn to do some programming. Program the robot and then use your program to make the robot follow the yellow path on the "*robot_maze*" background.

TASK: Load the code and then change it to make the robot walk more quickly.



For an example of this script in action, open RPiScratch/Projects/robot_v2.

Tip...

This is useful code for controlling characters in games.

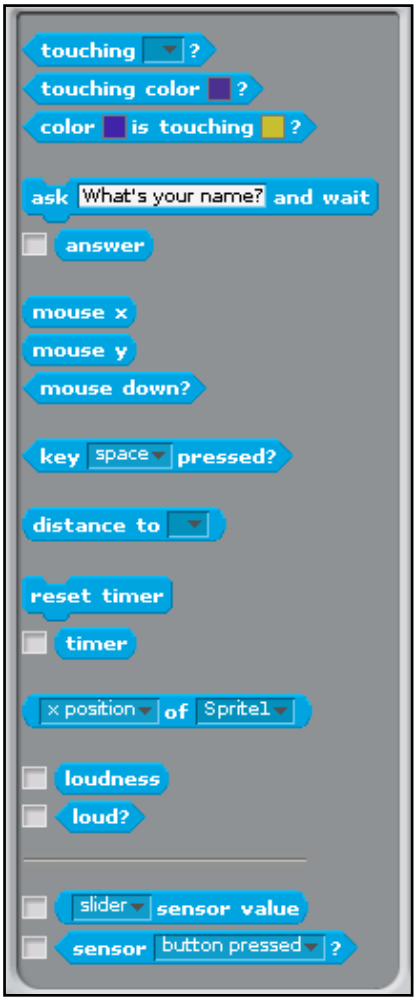
Line-following vehicle (LFV)

Notes:

RESOURCES: The sprite "*yellow_car*" and the background "*line_background*".

That robot is like something out of a corny science-fiction film. Let's look at a more modern example. Scientists are experimenting with vehicles that can drive themselves. One way of doing this is by programming the vehicle to follow a line painted on the road surface. These vehicles are known as "line-following vehicles", or LFVs.

In this exercise, we'll look in more detail at the blocks in the Sensing section.



The Sensing script blocks allow your program to sense changes in its environment, either in response to user input or events in the program.

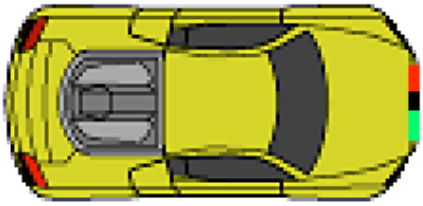
Our task is to make a car to follow a line, just like the real engineers designing LFVs for use on the roads. Our main tool is going to be the "color [ ] is touching [ ]?" block from the Sensing menu.

This block allows a program to detect when an area of one colour touches against another colour: for instance, as in the screenshot above, a patch of red touches a patch of black.

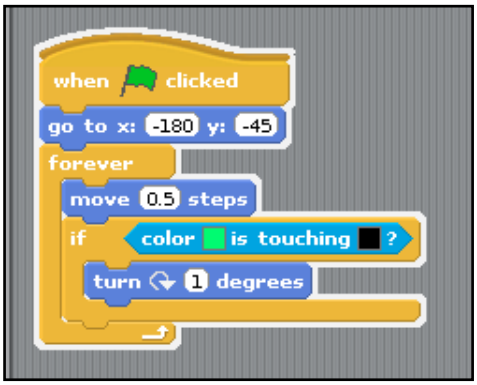
Once the program can detect two colours touching, then we can tell it how to respond when this happens. The colour event becomes a trigger, causing the program to do something.

To set the colours in your block, click on one of the little coloured squares. Your mouse-pointer will turn into an "eye-dropper". Use the eye-dropper to click on the colours you want to use in your block.

But how can we use the ability to detect colour to help make our LFV follow a line on the road? Well, let's start by importing our "*line_background*" and the sprite "*yellow_car*".



Look closely at the car, and you'll see that there's a patch of green on its bumper. We're going to use that green as our first "sensor". Build the script that you can see in the next screenshot.



Now, click on the green flag and see what happens. Oh dear, everything is fine as long as the line curves to the left but as soon as it curves to the right the car wanders off. What's going on?

The problem is that the car needs two sensors. Currently, if the line bends to the right, the black line will touch the green sensor – the program will detect this and tell the car to turn to the right.

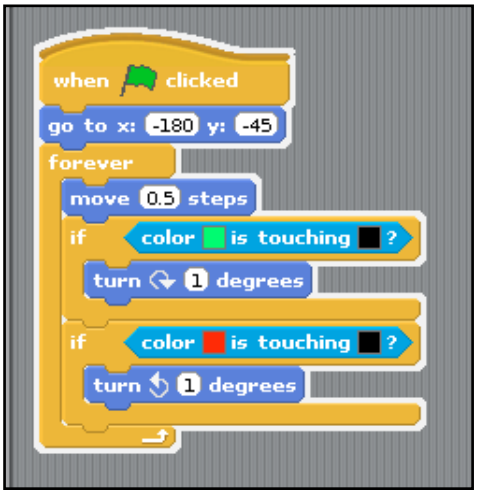
Notes:

Tip...

If you can't stop your car for long enough to sample a colour, hit the red "stop scripts" button at the top-right of the screen.

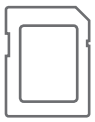
Now, we need to use the red patch on the car's bumper as a red sensor. The program will then be able to detect when the line turns left and tell the car to turn to the left too.

Here's the code you need to finish your LFV program:



As you can see, the program is run with a green flag event. After that:

- A "go to" block is used to position the car on the line.
- We the have used a "forever" loop to keep the car running.
- The car moves 0.5 steps (for each cycle of the loop).
- An "if" block is then used to check if the green spot on the car has touched the black line. If it has, the car will turn to the right by 1 degree.
- Another "if" block is then used to check if the red spot on the car has touched the black line. If it has, the car will turn to the left by 1 degree.



For an example of a completed version of this project, open RPiScratch/Projects/lfv.

Tip...

If your car stops at an odd angle, and you can't straighten it out, you can import the "yellow_car" sprite again, drag and drop your finished script onto the new sprite and then delete the old one.

Tip...

Run the program in full-screen mode – it is more impressive. Click on the right-hand button at the top-right of the Scratch window, just above the green flag and the red circle.

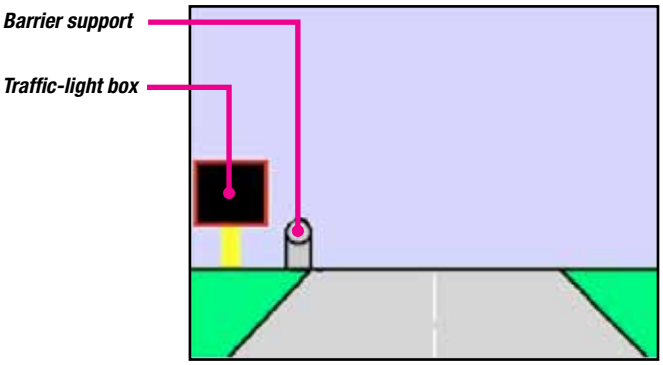
The level crossing

RESOURCES: The sprites "*lights*" and "*barrier*", the background "*level_crossing*" and the sound "*level_crossing_a*".

Computer control systems can save lives. At a level crossing we have to make sure that cars and trains never meet. This is done with a sensor on the track that detects the passing train.

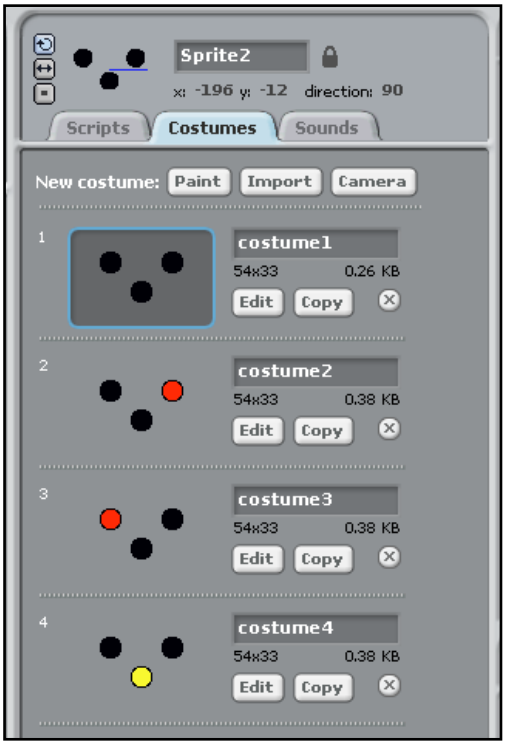
A computer then responds to this event by stopping the traffic approaching the level crossing and closing the barrier. This control program can be written in Scratch. But before we start, we need to import the necessary sprites and background.

First, import the background "*level_crossing*". As you can see from the picture on the next page, the background shows the road leading up to the crossing, the traffic-light box and the barrier support.



Notes:

Now we need our sprites. Import "*lights*" and "*barrier*" from the RPiScratch/Resources/Costumes folder.



Look at the image above, and you'll see that the "*lights*" sprite has four different costumes. By switching costumes quickly, we can make it look as if the lights are flashing.

The last thing we need to do before we start building our script is to import the sound of the level-crossing alarm. To do this, click the Sounds tab, find the sound "*level_crossing_a*" and then use the import function, just as you did with the background.

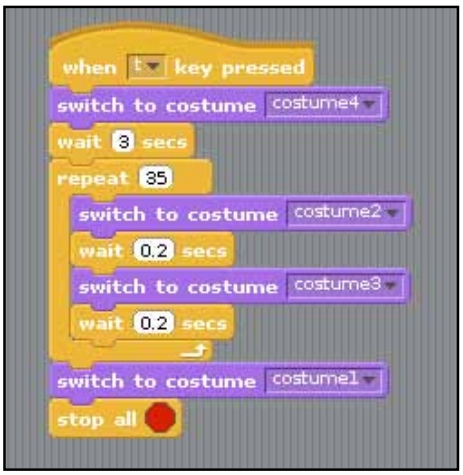
Import your sound effect from the Sounds tab. You'll find it right next to the Scripts and Costumes tabs.

Now, let's build our script. Before you start, select the "*lights*" sprite in the Sprites List.

First things first: we want to start with the lights out. So we use a green flag event to switch our lights sprite to "*costume1*", which shows three black lights.

As the train approaches the level crossing barrier, I have used an "when [ ] key pressed" block from Control to detect it, setting the value to "t". This event will trigger the lights, alarm and barrier. But first, let's deal with the alarm.

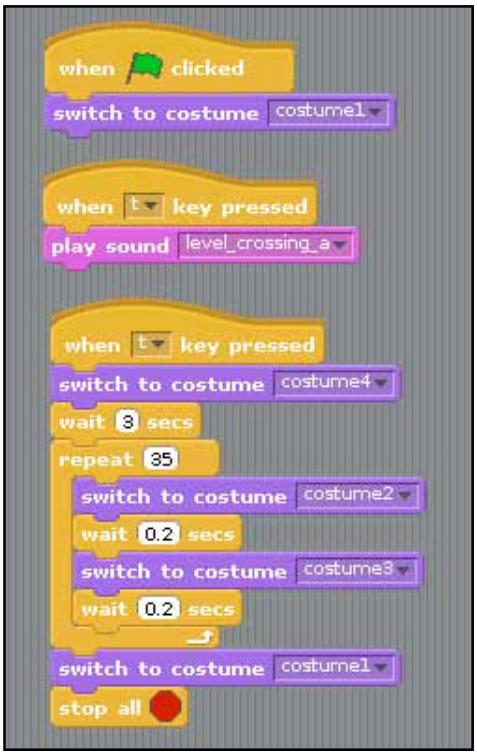
Build the script you can see in the screenshot above. This will cause the alarm to sound when the "t" key is pressed.



Notes:

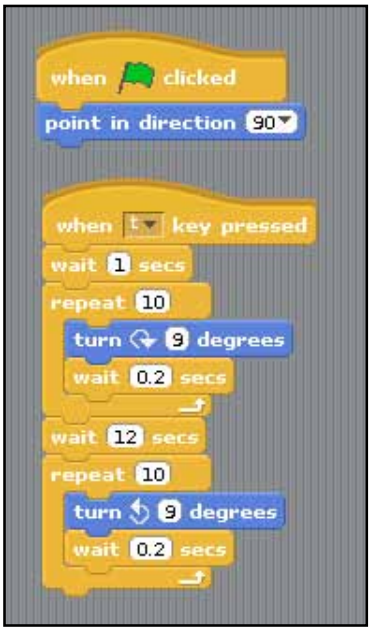
To make the red lights appear to flash we must switch the two red light costumes repeatedly. We use a "repeat" loop for this. Build the script on the previous page. You'll need to experiment with the number of repeats until you get the timing right.

Notes:



This is the finished program for the lights, with all the blocks in place. Notice that you have three separate scripts, not touching each other.

The barrier

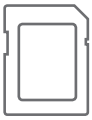


Now for the barrier: drag it until it sits on the barrier support, with its thicker end just resting on the top of the post. Each time the program starts, the barrier must be pointing upwards. Using a "point in direction [90]" block (from Motion) will take care of this.

In the script for the barrier, I have used a "when [ ] key pressed" block from Control, with its value set to "t". Think back – the script for the lights also starts when you press the "t" key. In this way, lights and barrier are synchronised: that means they both start at the same time.

To lower the barrier, we will rotate the pole in stages by using a "repeat" block. Experiment with the number of repeats and the angle of turn to move the barrier just the right amount.

After a set time, 13 seconds, we raise the barrier using another repeat block. The train has passed, all the cars are safe. Well done!



For an example of this project in its finished state, open RPiScratch/Projects/level_crossing.

Over to you

TASK: As an extra feature for this program, create a train sprite that moves across the screen when the barrier drops.

Lesson 1.7: Scratch games

LEARNING OBJECTIVE: In this exercise, you will learn some of the techniques used for game programming by playing some Scratch games.

Some of the most fun that you can have with Scratch is through programming games. You can make games about anything you like. We have included two sample games for you to investigate, and there are plenty more games and other examples of programs in the Scratch Projects folder.

The starting point for each new game is the theme or idea, and the aim of the game. When you have an idea for a game, think about the "story" you want to tell and the game's characters, players, pieces etc. that you will need. These will be the sprites.

You can find great images for sprites and backgrounds by importing them from the Scratch Media folders or by searching on Google Images, saving and importing into Scratch. If you want to get some idea of how to get started and what you can do, have a look at some other people's games. The best way to learn how to program is to examine other people's scripts to see how they made cool things happen.

Finally, you will need backgrounds for your stage to create the rooms, levels and scenes in your game, so that your characters have somewhere to move around in.

Notes:

Tip...

Don't forget to add sounds and music, as this greatly enhances a game.

Prancing Pony game

RESOURCES: The sprites: "*pony*" and "*girl*" and the background "*field*".
THE AIM OF THE GAME: A pony moves around a field, making random changes of direction. Using the mouse, the player controls the girl sprite. To score points, you must put the girl on the pony.

There are four tasks for you to program:

1. Move the pony round the field.
2. Make random changes in direction of the pony.
3. Move the girl with the mouse pointer.
4. Add a point to the score each time the girl is placed on the pony.

So, let's get to it!

Moving the pony

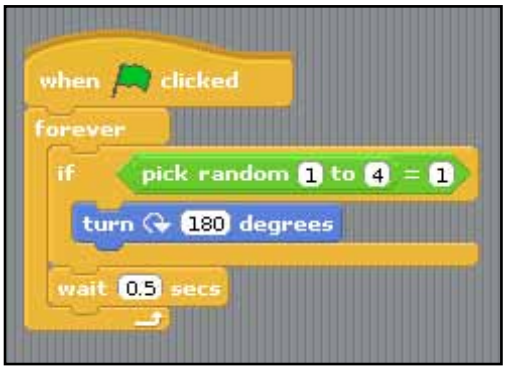
1. Keeping the pony moving

Select the pony sprite and drag a green-flag event to the Scripts tab. We will use the green flag event to run all the scripts in this game at the same time.

We have used a "forever" loop to keep our pony moving (remember your LFV). For each cycle of the loop, the pony moves forward 3 steps. If it hits the edge of the screen it will bounce off in the other direction; this prevents it from getting stuck.

Notes:

2. Random changes in direction



If the pony never changes direction except when it hits an edge, the game will get a bit dull. So we'll use another "forever" loop to make the pony randomly change direction from time to time.

The "pick random [ ] to [ ]" Operator block has been set to pick a random number from 1 to 4. If the number picked is 1, the pony will turn 180 degrees.

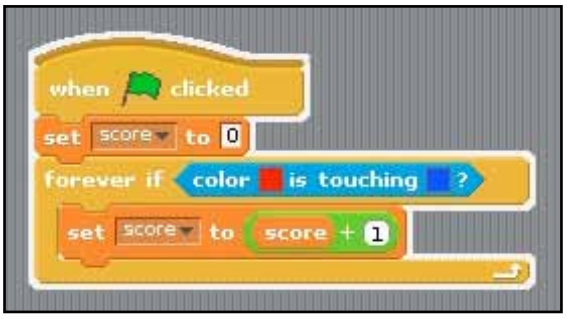
This is set to happen every 0.5 seconds by the "wait" block.

3. Move the girl with your mouse pointer

Select the girl sprite in the Sprites List. Using yet another "forever" loop, as well as blocks from Motion and Sensing:

- Set the x coordinate of the girl sprite to the x position of the mouse pointer.
- Set the y coordinate of the girl sprite to the y position of the mouse pointer.

4. Score a point each time the girl is placed on the pony.



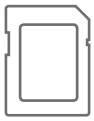
In this script, you can see the following:

- We have created a variable to handle the score and named it "score" (imaginatively!).
- When the game starts the score is set to "0".
- The "forever if" loop checks to see if the red colour of the girl's top is touching a small blue dot on the saddle of the pony.
- Every time the red touches the blue the score is changed to "score + 1".

Notes:

Hang on a second! Every time the red touches the blue dot, "score" is changed to "score + 1". If you are good at maths that should set the alarm bells ringing! How can something be itself plus one?

Don't worry; programmers often do this. Remember that a variable is not a number; it is a container that can store numbers. We can do a sum with the number and put the result back into the same container.



To see a working example of the Prancing Pony game, open RPiScratch/Projects/prancing_pony.

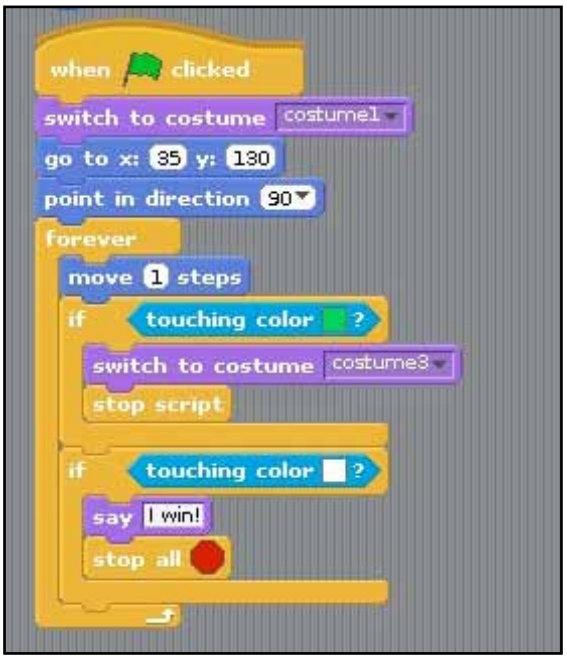
Racing game

RESOURCES: The sprite "*red_car*" and the background "*track*".
THE AIM OF THE GAME: A two-player game. Each player controls a racing car around a track. If the car goes off the track, it crashes. The first one over the line is the winner.

There are five tasks to program for each car:

1. Line cars up on the starting line.
2. Set the cars in motion.
3. Crash the car if it leaves the track.
4. Declare the first car over the line as the winner.
5. Oh, yes – and the players have to be able to control the cars!

1. Moving the red car



Start off with a green-flag event. When the flag is clicked, the car to should:

- switch to "*costume1*"
- position itself at the coordinates x: 35 y: 130
- point itself to the right (direction 90)

Tip...

Remember, to stop everything moving so that you can sample the colour of girl's top with the eye-dropper tool, hit the red "stop scripts" button. If you can't find the girl sprite, have a look in the bottom-left corner. She may have run off to hide there when you dragged your mouse to the Script tab.

Notes:

Notes:

To do all this, you'll need, along with the green flag, blocks from Looks and Motion. To move the car, we use our old friend the "forever" loop. The car moves one step each cycle.

If the car leaves the track, it will touch the green grass. This event will cause the first "if" block to crash the car and stop the script for the car.

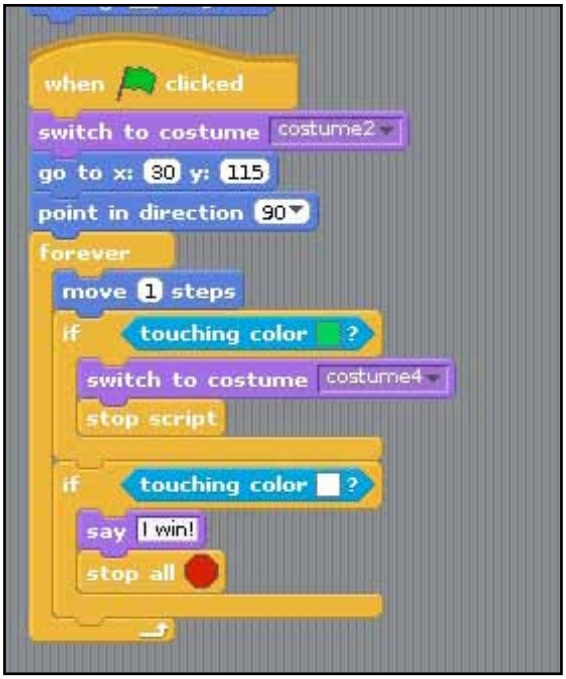
As the car crosses the finish line, it will touch the white line. This event will cause the second "if" block to say "I win!" and stop all the script running for both cars.

2. Controlling the red car

Ok, so we can get the red car moving. But we need to be able to steer it. To do this we need two "when [ ] key pressed" (from Control) events are used to control the car.

As you'd expect, the "right arrow" key turns the car to the right and the "left arrow" key turns the car to the left.

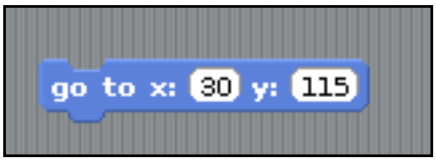
3. Do the same for the purple car



Similar code is used to control the purple car with some slight alterations. The easiest way to do this is to make the red car and then duplicate the sprite. Remember to change the costumes of your new car from red to purple.

The changes you need to make to your code for the purple car are:

The purple car should line up next to the red car at x: 30 y: 115.



You should also use the “x” key to turn the car to the right and the “z” key to turn the car to the left.

To see a finished example of this game, open RPiScratch/Projects/racing_game_v1.

Over to you

TASK: Add the sound of racing cars to enhance the game. You’ll need to record or create your own sound file.
TASK: Try to make these improvements to the racing game:

In the racing game you just created, the car says “I win!” when it hits the line. But the speech bubble is gone so fast, it’s hard to read. However, if we change the script so that the speech bubble stays on screen for, say, 2 seconds, then there’s a chance that the second car will also hit the finish line in that time and also say “I win!”. Then we wouldn’t know who had really won.

Try and create a new version of the game, saving it as something like “racing_game_v2”, this time causing the losing car to stop in its tracks the moment the winner crosses the finish line. To do this, you’ll need to use a variable.

Call this new variable something like “have_winner”. The default value of “have_winner” should be “0”. Whenever either car wins, by touching the white finish line, it should set the value of “have_winner” to “1”. This should be the trigger for both cars to stop moving.

Finally, instead of having the cars crash when they hit the grass, try and make them slow down, so that the player has a chance to get them back onto the track.

To see a finished example of this game, open RPiScratch/Projects/racing_game_v2.

Notes:

Tip...

When you are creating a game, don’t be over-ambitious at the beginning. Program your basic idea and make sure that it works. Then develop your game bit by bit, adding new characters, events and levels. It is a good idea to save each new version with a different number, so that you can keep track of the changes that you have made. You may get stuck and want to go back to a previous version of your game.

What next?

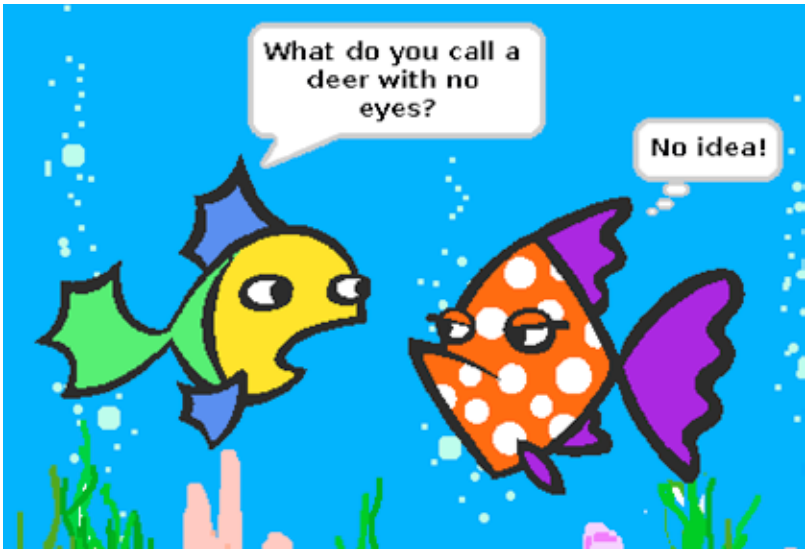
Congratulations! You are now a Scratch programmer. Hopefully, this is just the beginning for you, and you will be inspired to create your own programs using the Scratch language. There are many other skills and programming techniques to learn.

Happily, one of the great things about Scratch is the wealth of support and advice on the internet. I have included a few links to some online tutorials for you to explore:

http://learnscratch.org
http://www.scratch.ie
http://scratch.redware.com/index.php
http://blogs.wsd1.org/etr/?p=395
http://scratched.media.mit.edu
http://morpheus.cc/ict/scratch/default.htm

And why not tell your teachers about Scratch? Maybe you and other like-minded kids from your school could even set up a Scratch Club. And when you have projects to do, you could ask your teachers if you can do them using Scratch. The most important thing is to have fun with Scratch.

Stuck and don't know how to get unstuck? Don't suffer in silence. There are some great, friendly forums on the Scratch website where there are lots of other Scratch users to help you.



Build your own blocks

Notes:

When you become an advanced user of Scratch, you may find that there isn't a script block for a particular job you want to do. Or you might want to improve your program by creating better blocks. Help is at hand with BYOB at ***http://byob.berkeley.edu***

Thank you for reading this guide. I hope it helps you to have hours of fun with your Raspberry Pi.

Now it's time to move on to the next chapter, in which you will learn how to program in the Python language as well. Exciting stuff!