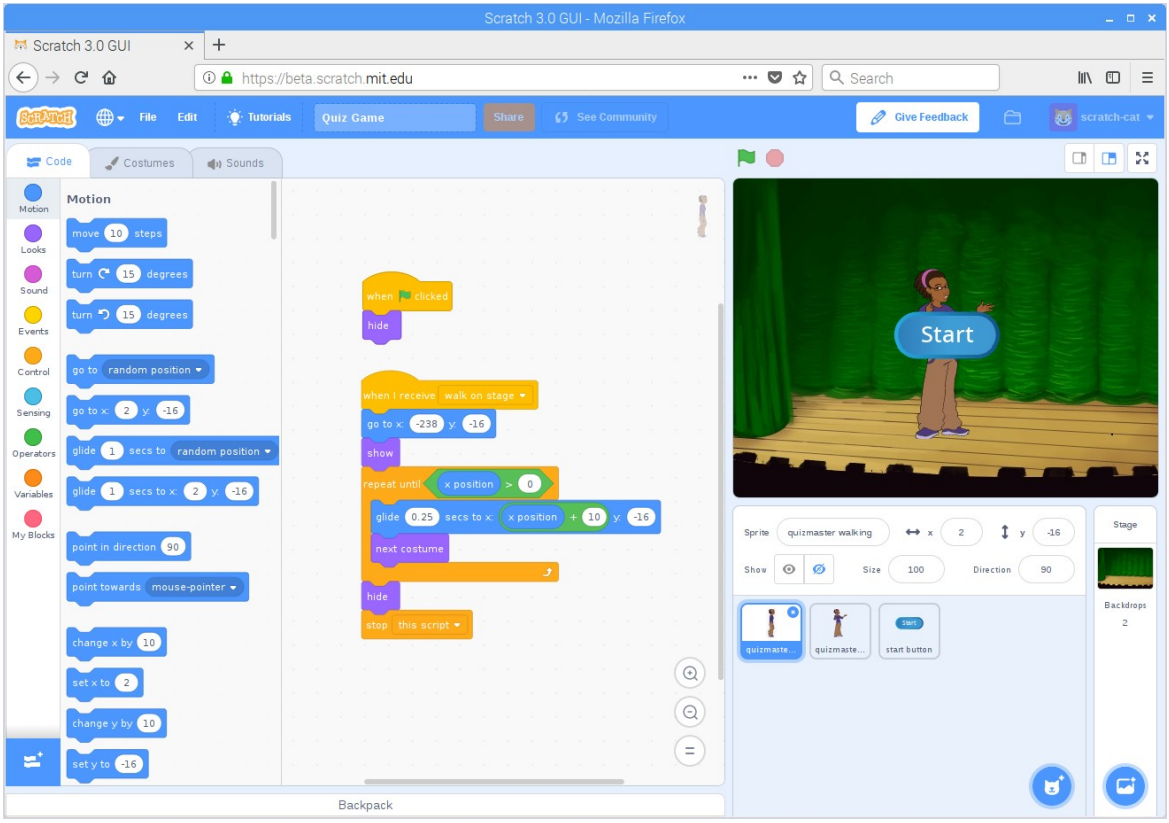
Crash Course in Scratch Programming

Quiz Game



For Scratch Version 3

Stewart Watkiss

PenguinTutor.com
www.penguintutor.com/programming

January 2019

About this guide

Scratch is a graphical programming language designed for children to learn computer programming (coding). It is block based with blocks of code that can be joined together to make a computer program. It creates a graphical computer program which is great for making games. Scratch can also be used to control the output pins on the Raspberry Pi which can be used as a way into physical computing.

This guide uses Scratch, but is intended for a slightly older audience as it introduces a lot of techniques quickly. This can be used by older children or adults wanting a quick crash-course in programming with Scratch.

Working through the guide will create a computer game and in doing so introduce all basic programming techniques. This includes creating a programming sequence (from design to code), selections (conditional if statements), repetition (repeat and forever loops). It also covers variables (including strings and lists) and the inputs and outputs.

This guide originally started out as a guide for teachers to accompany a training session I provided in schools. It has been updated to the current version of Scratch (version 3) and updated to appeal to a wider audience.

The light-bulb is used to indicate that this is additional information or a suggestion for future development.

About the author

Stewart Watkiss has a master's degree in electronic engineering from The University of Hull and a master's degree in computer science from Georgia Institute of Technology. He has been working in the IT industry for the over 20 years, including working with computer networking, Linux and IT security.

As a STEM Ambassador Stewart enjoys teaching programming with Code Club and Raspberry Jam events. He is author of the book “Learn Electronics with Raspberry Pi” and runs the website www.penguintutor.com

Table of Contents

Introduction

Following this guide will create a computer quiz game using the Scratch programming languages. It is recommended that you follow this guide by recreating the program yourself in Scratch. This will provide an opportunity to try the different techniques directly and will result in a complete working game. There are then suggestions for improving the game which can be used to expand the game further and test understanding of the key concepts.

Full details of the project are available from the accompanying website www.penguintutor.com/programming . This includes a link to the Scratch website where a copy of the completed game can be found.

About Scratch

Scratch is a programming language created by the Lifelong Kindergarten group at MIT. It is designed for teaching programming to children aged between 8 and 16, but can be used by people of all ages. It is normally run as a web-based application through a web browser[1], but there are also offline versions.

Scratch is usually considered the first choice when looking at a PC based programming language for teaching primary school children. When used with the Raspberry Pi it can also be used to interface with simple electronic circuits as a first step in physical computing.

The latest version is Scratch version 3 which is primarily web-based and accessed through a web browser using the HTML 5. This is an improvement on the older versions which were reliant on the proprietary flash plug-in.

Scratch website: scratch.mit.edu/

Getting started with Scratch

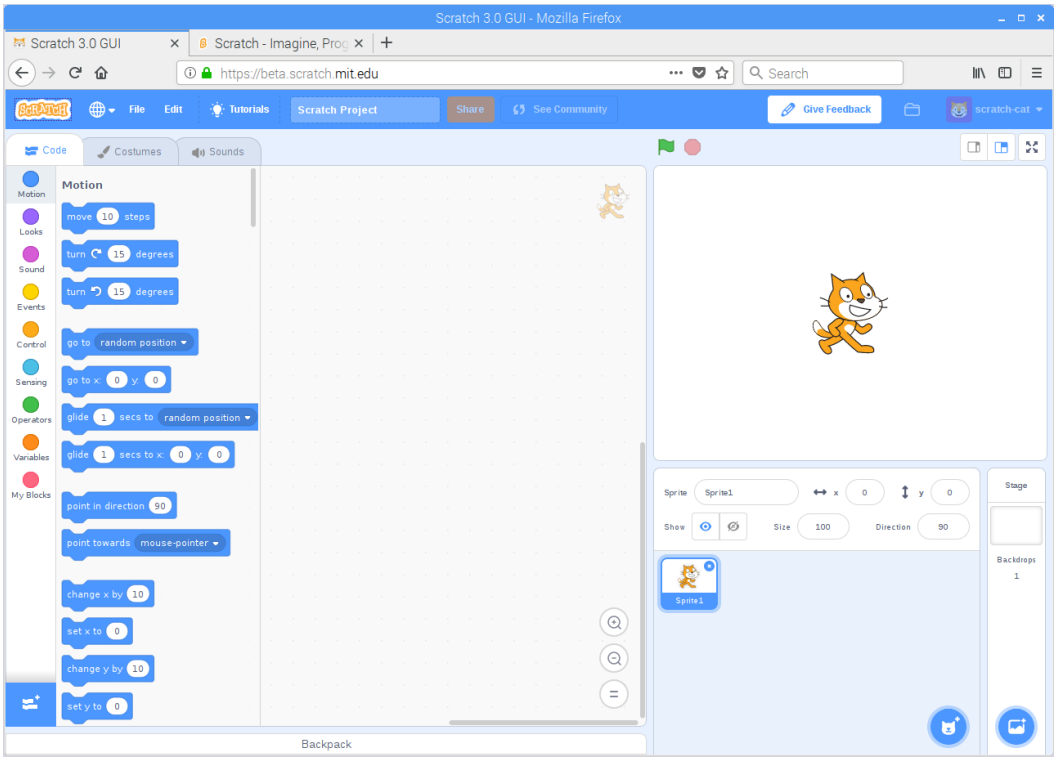
Scratch uses a drag-and-drop approach to programming making it very easy to use. It comes with a development environment that can be used to create programs without needing to learn any complicated rules about the computer language.

You can start with Scratch by going to

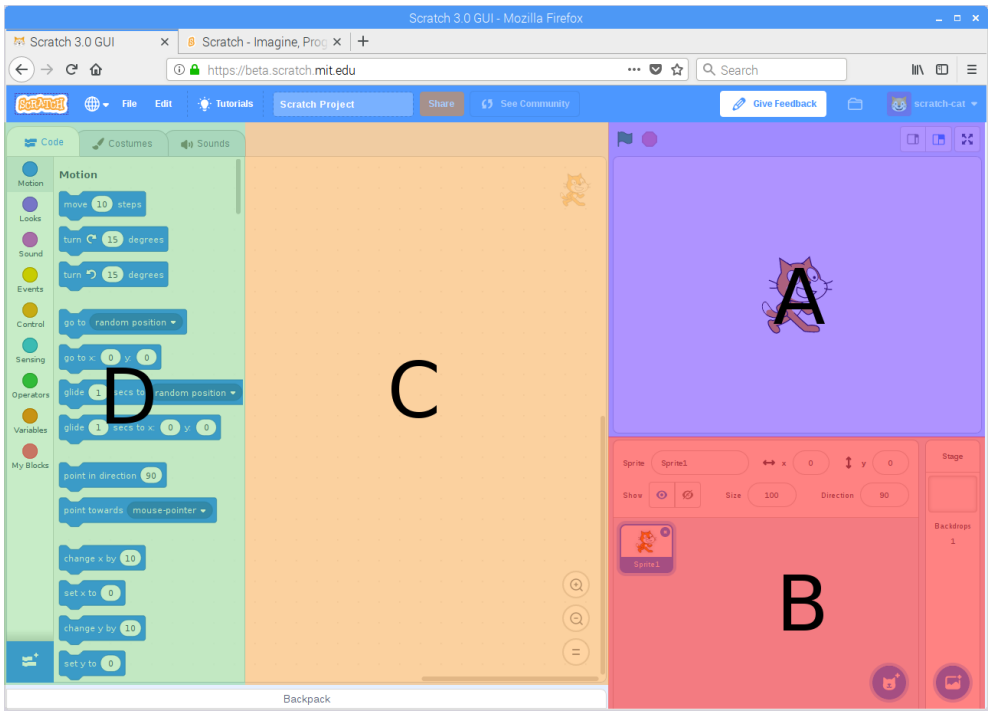
scratch.mit.edu

1 Scratch works with Firefox, Chrome, Safari and other browsers using the same toolkit. It does not work with Internet Explorer.

Click on the Create option to get started which will place you in the Scratch editor.



The application is split into four main sections, which I have coloured and labelled below as A, B, C and D to make them easier to identify.



A. The Scratch stage

The most important part of the Scratch application is the stage area (section A), this is a representation of the screen when the program is running. The look of the game will be designed on the stage area and it's also where you can see the program run. You can enlarge the stage to fill the screen using the small icons to the top right of that section.

There is a Green Flag, which is used to start the program running and a Red Stop Sign, used to stop the program.

B. The sprite list area

If the screen is a stage then the sprites are the actors, props and special effects used to create the show. In Scratch a sprite can have it's own costumes, code and sounds associated with it.

New sprites are created in the sprite list area (section B) where they can then be loaded into the sprite editing area.

The stage is considered a special type of sprite and is always shown in the right part of the sprites list area. The stage can contain multiple backdrops used to change the image shown on the background of the stage.

C. The sprite edit area

When we want to change a sprite then we load it into the sprite edit area (section C). Sprites are

loaded into this section by click on them in the sprite list, where they will then be shown with a border. There are three tabs in the sprite area called Code, Costumes and Sounds.

In Scratch the program is grouped into code scripts which are accessed through the Code tab. It is fairly common to have several scripts of code associated with each sprite. The stage can also have its own code which is often used to control the sprites or the stage background.

The costumes tab is used to change the look of a sprite. It does not need to be a different form of clothing (although that is one use of a costume), it could be used to represent a different object or completely different look (such as a firework rocket which changes to a burst of light when it explodes). When editing the stage the costumes tab is replaced with a Backdrops tab which can be used to change the scene.

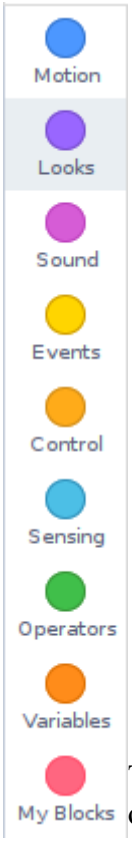
The sound tab is used to add sounds to the sprite. We won't be using sound in this example, but it is something you may like to add later.

D. The code blocks area

The code blocks area (D) holds the blocks of code that can be built together to make the program. Each block of code is shaped with interlocking tabs which can be connected with other appropriate blocks of code.

This area will only show when the code tab is selected.

The blocks of code are grouped into nine different groups which are colour coded. In version 3 of Scratch you can access the different types of code block by scrolling through the code blocks, or you can jump straight to that part by clicking on the colour circle on the left.



Motion – moving a sprite around the screen.

Looks – appearance of the sprite. Also used to represent bubble speech.

Sound – make noises and play music

Events – blocks that indicate when the code should start running

Control – used to control the flow of the code and handle decisions

Sensing – detect the position of the sprite and the mouse

Operators – perform logical and mathematical operations

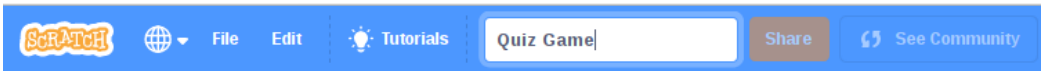
Variables – variables which can be used to hold information such as numbers or text

My blocks – your own custom blocks

There is also a menu bar across the top of the screen, along with the usual menu options the menu bar is useful for naming and sharing the project or downloading a

copy to your local computer.

Before starting making the game it's a good idea to give the project a name. As long as you are logged in the game will then be saved as you work through it.



There is also a backpack at the bottom of the screen which can be used to share items between projects (not covered in this guide).

Designing a game

Before a game is created in Scratch it is useful to design the program on paper first. This could be a simple page describing the way the game will work or something more involved including example pictures and diagrams.

I have kept the paper design fairly simple as an explanation of the game we are going to create and some of the key code and variables. I have called this the Design Specification, although it does not need be treated as something quite so formal.

Design Specification – Quiz Game

Aim

This is a quiz game. After clicking start a quiz master will come on to the stage and ask 3 questions. The player will respond with a string answer, and will be told if the answer is correct. At the end the player will be told how well they did. The game can then start again using the start button.

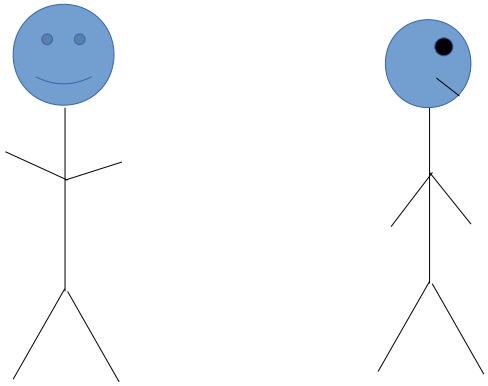
Sprites

A list of sprites to be used in the game.

StartButton - "Click to start"

QuizMaster – Girl or Boy
Standing (facing front)
Walking (facing side)

These are stick drawings, built-in sprite costumes will be used in the actual game:



Variables

It is not necessary to list all variables, only the main ones and those shared between different scripts.

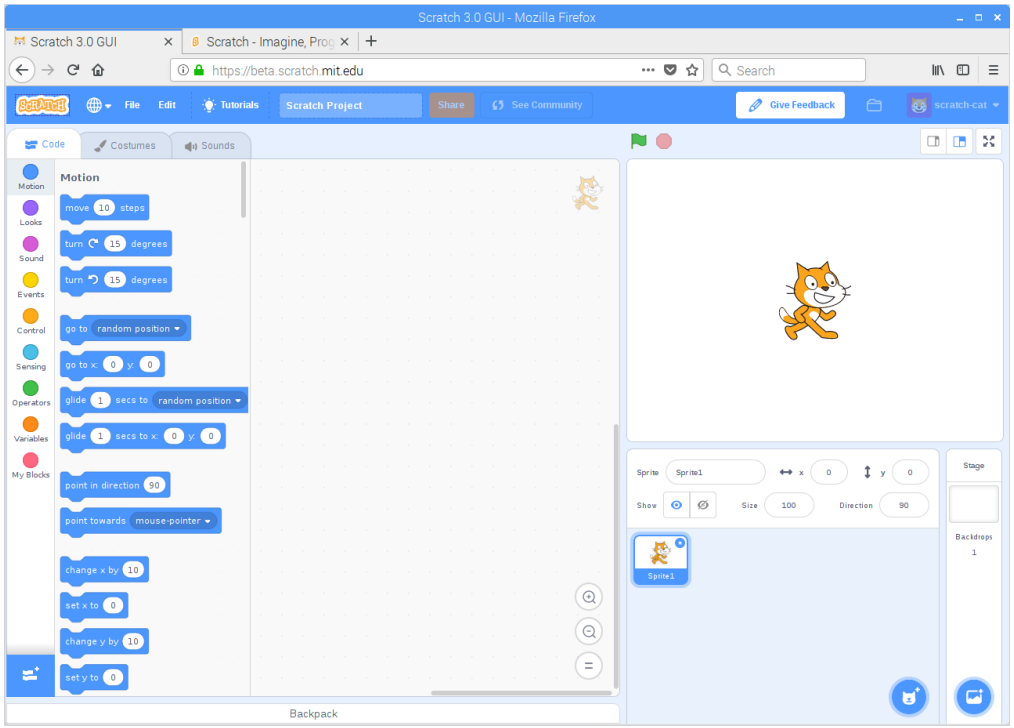
score – number for the current score
questions – List of all questions
answers – List of all answers

Creating the code

The order of creating the game comes down to personal preference. For this game I started by creating the sprites first which means that I could set-up the visual aspects (sprites and costumes) first and then concentrate on the code afterwards.

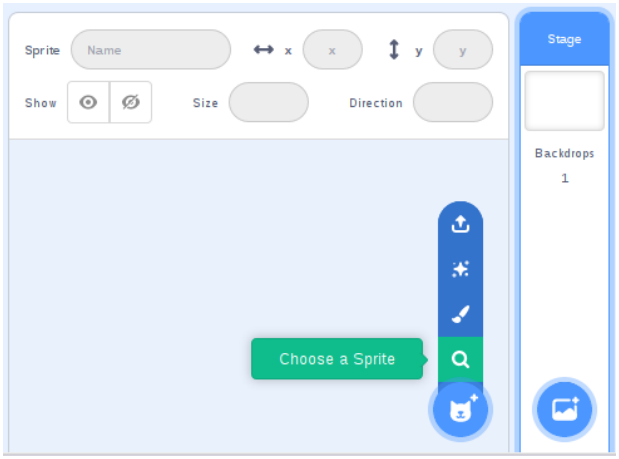
Setting up the Sprites

When you launch Scratch you should have a blank program with a cat sprite shown on the stage. This is the default setting for new programs.

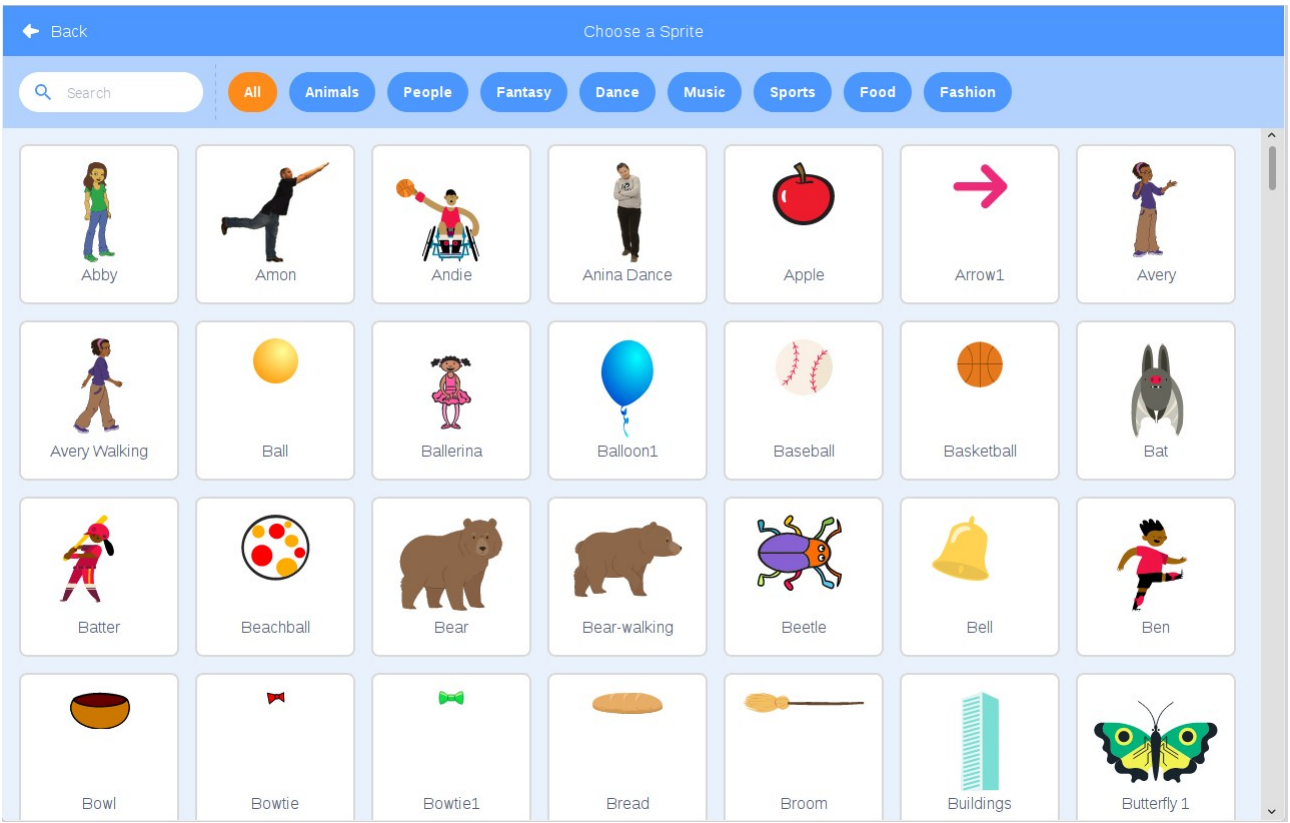


You won't need the cat so delete Sprite1 by right clicking on the Sprite in the palette and clicking delete.

Next add an image of a person who will be our quiz master. Create a new sprite using the Choose a Sprite option (shown in the image below). If you preferred you could create your own image or load one from your computer.

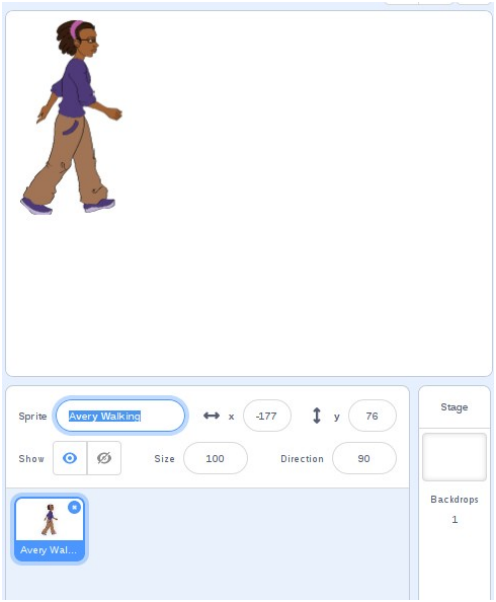


You can then select an appropriate sprite from the library. Many of the sprites have different costumes which you can preview by hovering your mouse cursor over the sprite. In this case I want to the sprite to start by walking on the stage, so I chose the "Avery Walking" sprite.

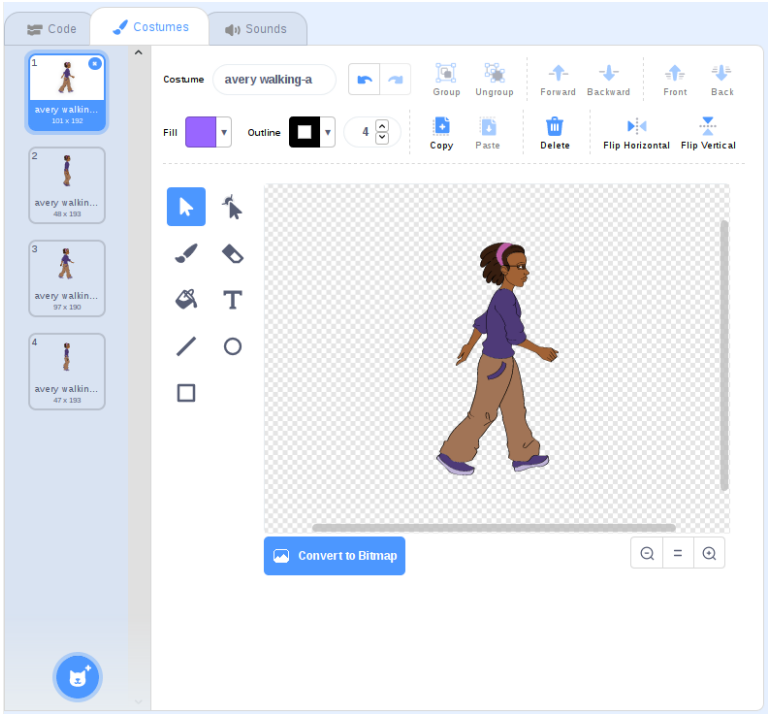


By default all the sprites are shown on the window. You can limit which are shown by

clicking on one of the filters at the top to only show that category.
The new sprite will appear in the sprite library and shown on the stage. When the new sprite is created it will be named after the name in the library. You can rename the sprite by selecting the sprite (it should have a border highlight) and changing the name to "**quizmaster walking**". It is a good idea to choose an appropriate name for the sprites as this can help in larger programs if there are multiple sprites.

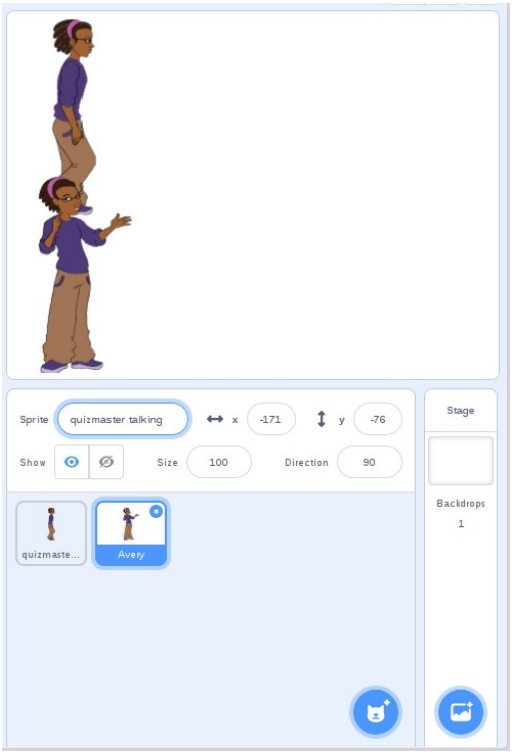


Clicking on the sprite will load it into the edit area on the right and choosing the costumes tab will show what costumes are available for this particular Sprite.



As you can see from the image above there are 4 different images which show the girls feet in different positions as though she is walking. By cycling through these it can make it look as though she is walking across the stage.

You will also need a costume with the girl facing forwards to ask the questions. It is possible to add a new costume to this sprite or create a new sprite. In this case you should create a new sprite by clicking on the Choose a Sprite button in the sprite area. This time choose Avery and rename to "**quizmaster talking**".



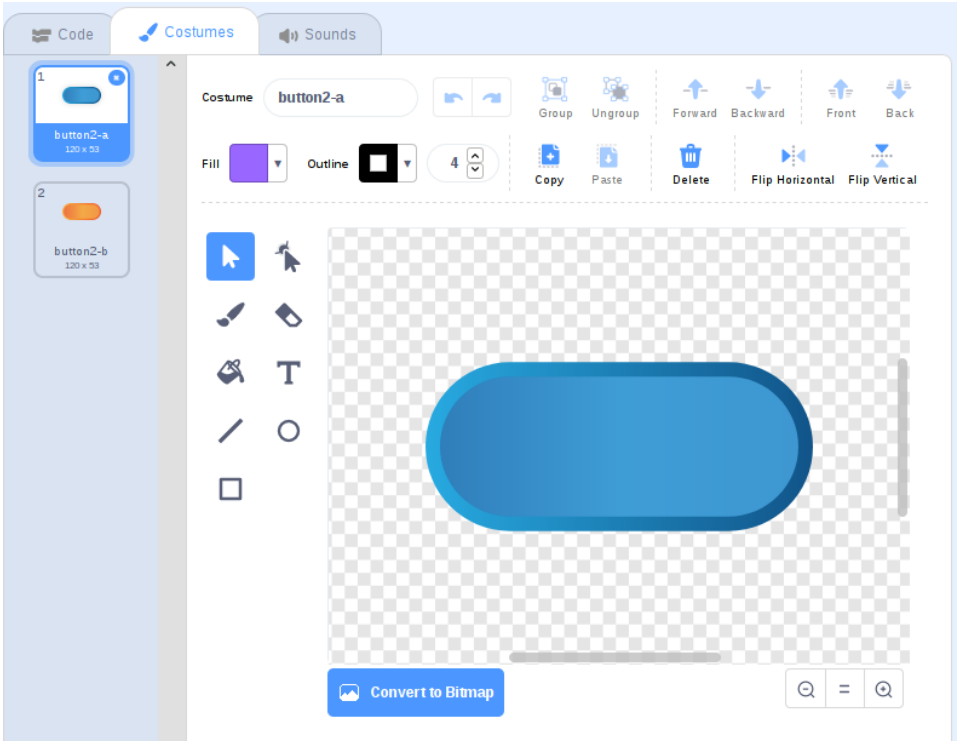
This shows both sprites on the screen at the same time. To hide the new sprite click on the eye with the line through under the sprite name field.

You will add the code to go with the two quizmaster sprites later. Before then add a start button as a new sprite.

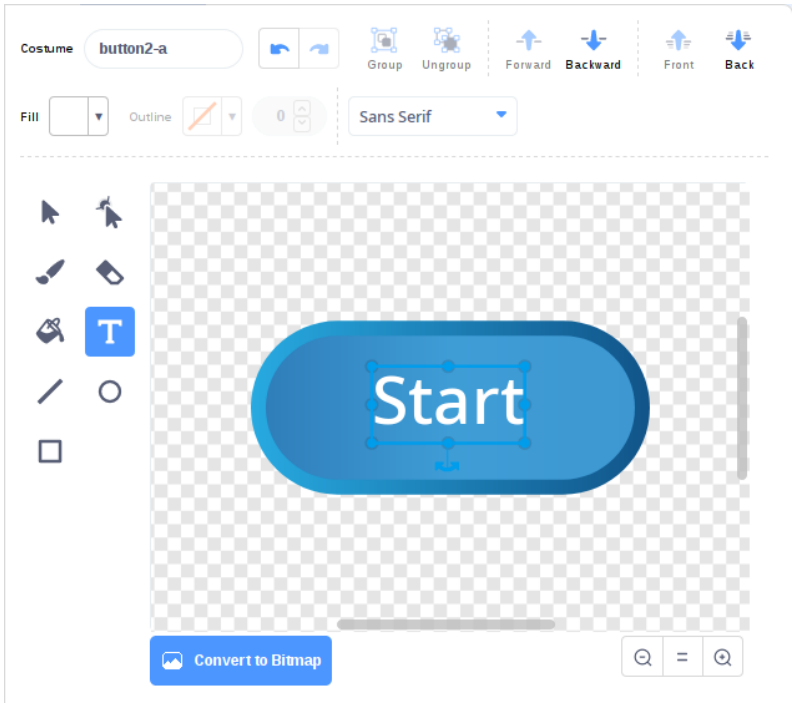
Use the "Choose a Sprite" option. This time search for button and click on either Button2 or Button3 which are the ones that have space for adding text.

You should position the sprite in the centre of the stage (if not already). Initially the button is plain with no text. We will need to edit the costume to manually add the text.

Rename the sprite to "**start button**" and click on the costumes tab.

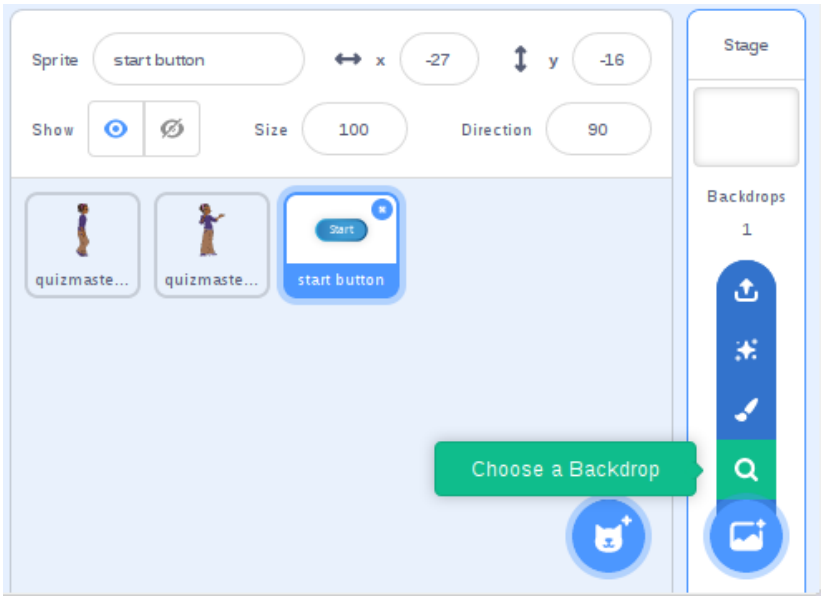


This shows an image editor where you can change the image. In this case select the T button to add text and write "Start" across the centre of the button.

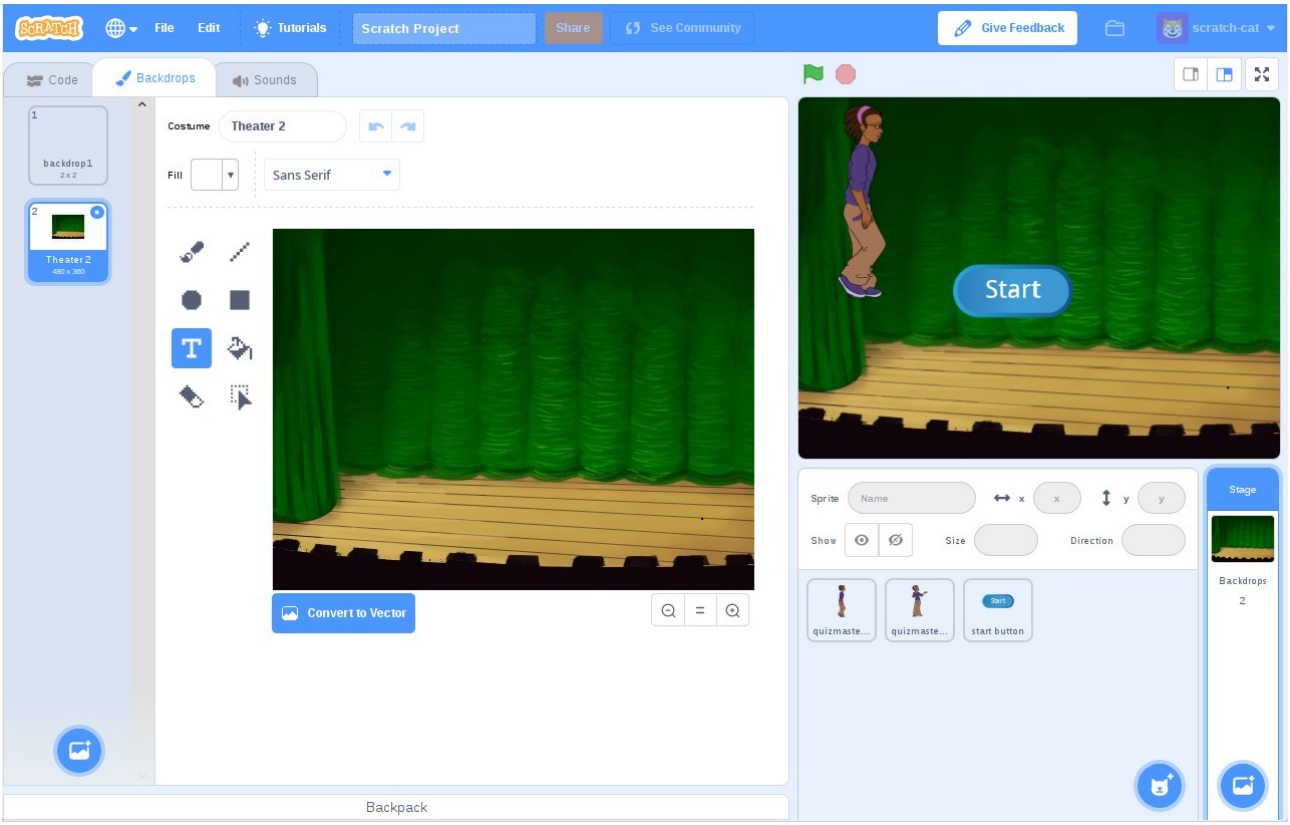


You can change the colour of the writing using the Fill option and change the size and position of the text by using the grab handles or dragging the text. You will then see the updated sprite on the stage.

The last you may like to do before creating the code is to add a better looking background. Click on the Choose a Backdrop button from the bottom right.



I have chosen the **Theater 2** backdrop which is stage suitable for a quiz show host.



You should now have three sprites "**quizmaster walking**", "**quizmaster talking**" and "**start button**". You will also have two Backdrops: the original blank screen and the Theater backdrop. You can delete the blank one if you like, but you won't need to change the backdrops in this project

so it doesn't matter if it is just left unused.

In this project you will have created all the sprites before adding the code, but in other projects you may prefer to add the sprites and the code at the same time.

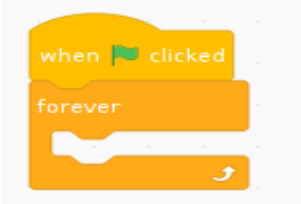
The main program code

When creating code in Scratch the code can be associated with a particular sprite, or on the stage. This will depend upon how the program needs to interact with the user. In this project there will be a master script which maintains the status of the program and that will communicate with other code that determines how each of the sprites behaves. This uses the concept of a state machine that changes state during the game. The main program (on the stage) will then send and receive messages with the code associated with the sprites.

In theory this main program could be associated with any of the sprites and that sprite would be responsible for triggering the other actions; placing it on the stage makes it easier to find particularly on larger games that may have dozens of sprites.

To create the first code, click on the stage Scripts tab and add the "When  (green flag) clicked" block to the scripts area by dragging it from the code blocks. This will mean that this script will start running when we click on the green flag.

Next add a forever loop below the when flagged clicked block by dragging it from the control category. The two blocks of code will snap together.

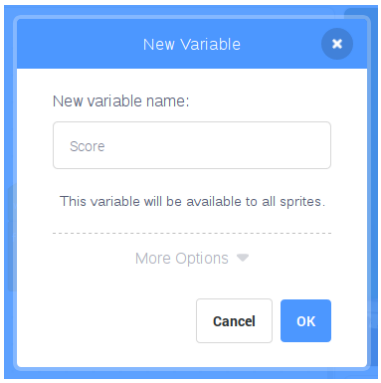


The forever loop will run the entire time that the program is running and anything that is included in that will be continually repeated. How often a loop is repeated depends upon the different steps that are called within the loop. In this case the forever loop will launch most of the other scripts and will run once for every time that the game is played.

If we didn't include the forever loop here then we will only run the quiz game once and it will then end. To play again then the player would need to start the game again using the "Green Flag". That would work too, but the forever loop is used here to show how a program can keep running after it would otherwise end.

Next you will need some-way for the computer to remember what the current score is. The way to store this is to create a variable. In Scratch a variable can hold different values such as numbers or strings of text.

Create a new variable by clicking on the Variables group and then click “Make a Variable”. Give the variable the name “**Score**”. As you are creating the variable whilst on the stage edit screen you will be told “This variable will be available to all sprites”, the alternative when creating variables for sprites is to have the variable only accessible from that sprite. You will add a variable restricted to a particular sprite later.



The variable will now be shown on the stage. If you don't want the variable displayed you can remove the tick mark next to the variable name, but you can leave the Score displayed on the stage.

You can then use the Variable code blocks to change that value of the variable. The variable code blocks have a small pull-down menu to determine what variable it applies to. Use the “set <variable> to 0” block and change variable to **Score**. Place that block as the first item in the forever loop. This will reset the game score to zero at the start of each run of the game.

You will need the code to show and hide the start button as required. The **stage** cannot control the Sprite directly so we need to send a broadcast message that tells the start button sprite to show and hide as appropriate.

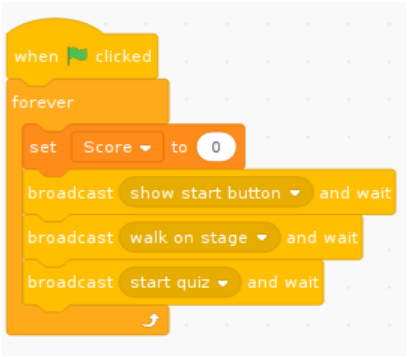
Add a “broadcast and wait” block (from the Events category) to the forever loop and create a new message called “**show start button**”; This is done using the small down-arrow and choose “New message”.

The code will also need to send a message to tell the **quizmaster walking** sprite to walk across the

stage. This can be done by adding a second broadcast and wait block with the message “walk on stage”.

You will also need to be able to communicate with the **quizmaster talking** sprite which can be done by creating another broadcast message that the **quizmaster talking** code can listen for. So add another broadcast and wait block with the message “**start quiz**”.

The script should now look as below:



This is effectively our state machine which defines different states that the program is in.

The first script complete, although it won't actually do anything until you create some code on the sprites to respond to the broadcast messages.

Before you move on to the next scripts here is an explanation of what this script will do. Read through the explanation and compare it with the code on the screen.

- When the green flag is clicked start a forever loop. The loop here means that when we reach the end of the quiz, instead of the program stopping it will start again.
- Set the **Score** variable to 0 – so far the player hasn't scored any points.
- Send a broadcast to tell the **start button** sprite to show itself and we then wait until it the **start button** script finishes. The script on the **start button** will run until the button has been clicked by the user, but we haven't created that script yet.
- When the **start button** script finishes then the program will continue to the next broadcast which is going to tell the **quizmaster walking** sprite that they should walk onto the stage.
- When the code on the **quizmaster walking** sprite finishes there will be another broadcast to tell the **quizmaster talking** sprite that it should start the quiz.
- When that script on the **quizmaster talking** sprite finishes (after all the questions have been answered) then it returns to the forever loop. As the program has reached the bottom of the loop it will jump back to the top of the forever loop and start again.

Note that whilst we have referred to the broadcast as going to a particular sprite. A broadcast is however seen by all the sprites including the stage and it's just about which of the sprites choose to

handle that broadcast message. In this case only one of the sprites will act on each of the broadcast messages but multiple sprites could listen for the same broadcast. If multiple sprites use the broadcast message then the script will wait until all have finished before continuing past the wait.

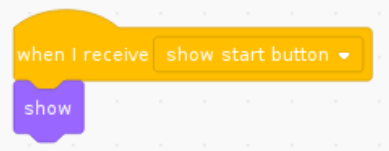
The start button

The **start button** will appear when the game is first run (in this case when we receive the broadcast message). Once the player clicks the button the code on the button will hide the button and allow the program to start.

The **start button** sprite is currently visible on the screen, however during the second run through the program the button will have been hidden. So the first thing that the **start button** code needs to do is to show the button.

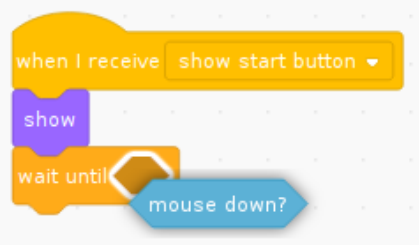
Click on the **start button** sprite so that it is highlighted in the sprites panel. The code area should now be blank.

Add the block "When I receive show start button" from the control blocks (you may need to select the appropriate broadcast message using the pull-down option on the code block). Then add to that a show block from the Looks code blocks. Ensure they snap together.



The program now needs to wait until the mouse is clicked on the button. This is done using the "wait until" block. As its name suggests the script will wait until the specified condition is met.

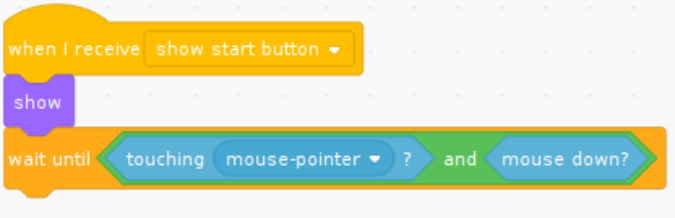
The block shows what looks like a stretched hexagon. This is where the condition needs to be inserted. If you look in the sensing code blocks they have the same kind of shape (but are stretched more) and can be dragged into the appropriate position in the wait until block.



This would work whenever the mouse was clicked, but it doesn't differentiate between whether the mouse was over the button or not.

To be able to check for both the mouse being over the button **and** the mouse button being clicked you need to use an “and” block (from the operators category), with “touching mouse-pointer” (from the Sensing category). The blocks are in reference to the current sprite so touching mouse-pointer will be true when the mouse pointer is touching the button, you then also need the mouse button to be pressed before the and condition is met.

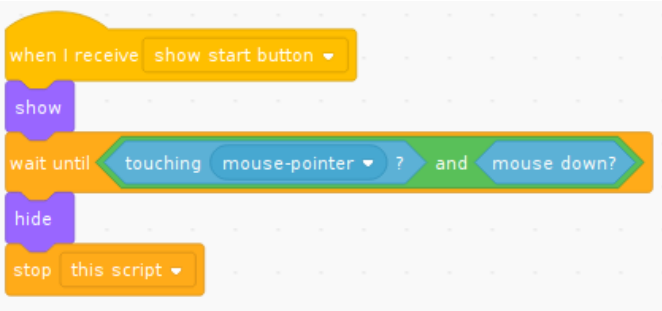
Remove the “mouse down” element if you had dropped that on previously. Drag the “and” code from the code blocks and place that into the wait until block. You will see two more stretched hexagons where you should then add the “touching mouse pointer” and “mouse down?” code blocks. When complete this should look like the code below.



If you are having trouble placing the blocks first make sure they are the correct shape (if the hole has angled ends then the code block to insert there must also have angled blocks). Then move the sensing code block so that the left hand-side of the sensing block is over the hole.

When the button is clicked the code needs to hide the button and stop this script so that the main script can continue.

The finished script should look as follows:



If you are struggling to find any of the code blocks, look in the same colour code blocks and also look for the pull-down option on the stop block.

You can test the code created so far by clicking on the green start flag. The start button should show, when clicked it should disappear briefly and then because we haven't created the quiz yet it should go to the next loop around and the button reappear. You should click the red stop button once you have tested the code.

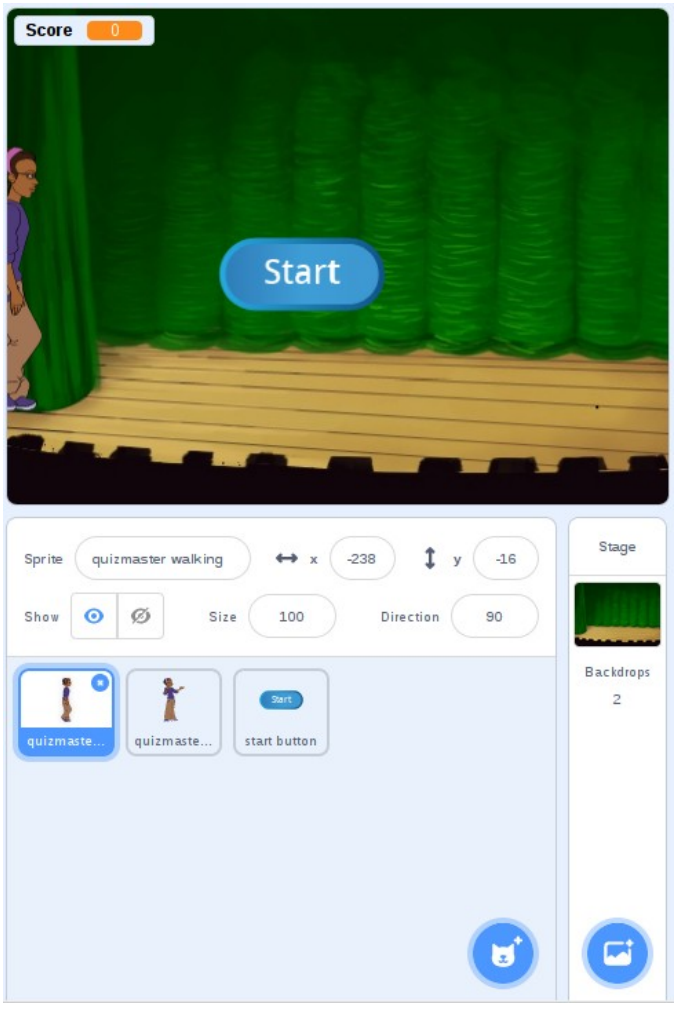
The “stop script” block is not actually required. When the code reaches the end of a script then it will stop that script automatically. It is however good practice to show the end of a particular script and helps to make it easier to understand. In this game I have added the stop script on the end of all the code that is called by the broadcast messages as the main script will be waiting on them to complete before continuing.

The quizmaster walks on the stage

The next broadcast is the one “walk on stage”. This is used to tell the **quizmaster walking** sprite that it should walk onto the stage.

Screen coordinates

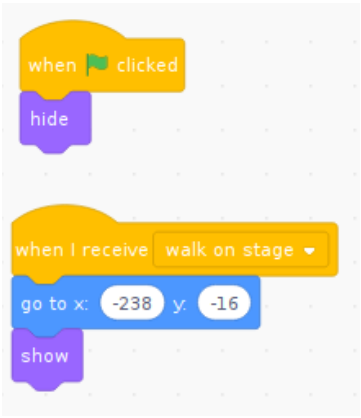
Click on the **quizmaster walking** sprite. Now move the **quizmaster walking** sprite so that it is just visible on the left hand side of the screen, but as though it is about to walk onto the stage. Note the positions of the x and y coordinates shown in the sprite properties (in my case x = -238 and y = -16). This will be the starting position of the sprite. You will want to hide the quizmaster walking sprite at the start (when waiting for the start button to be pressed) and for it then to appear in that position.



Create the following script which will hide the sprite when the program is started.

This code will only run once when the program is first started.

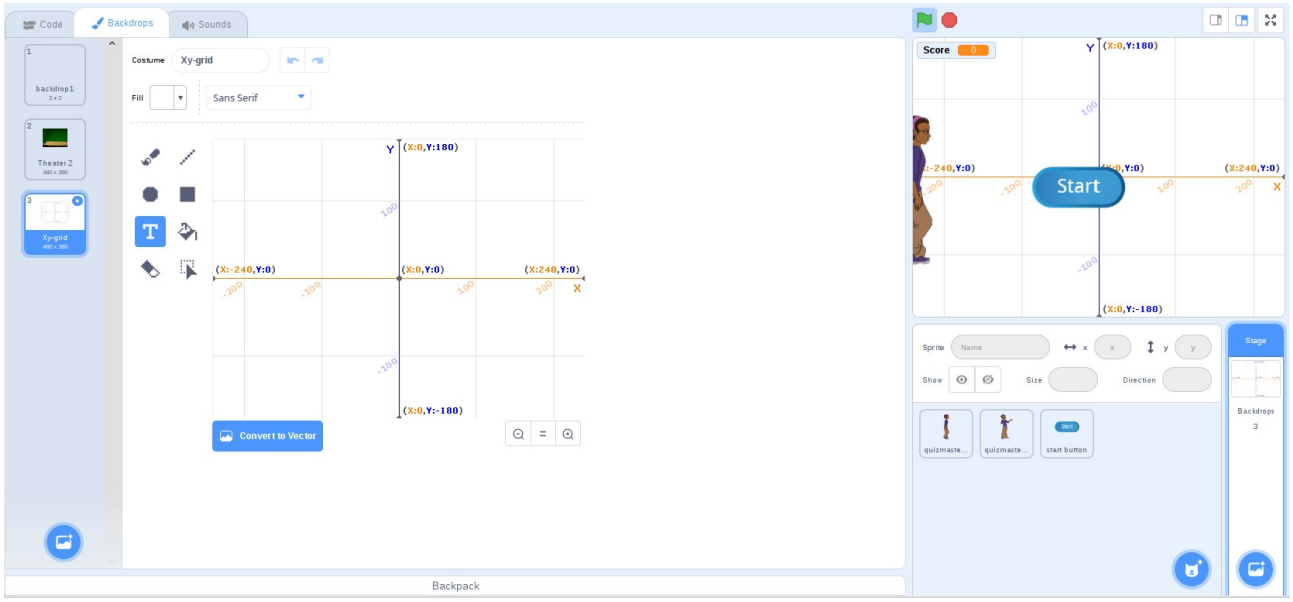
Next create a separate block of code (in the same code area, but not connected to the previous block) which listens for the “walk on stage” broadcast message, moves the sprite to the position on the left of the screen (using the coordinates noted previously) and then shows the sprite.



The values in the “go to” block should match the current position, or you can change them by clicking on the white blocks.

At this point we can take a small digression to understand how the grid system works in Scratch.

There is a backdrop in the library called Xy-grid. There is no need to add this to the game, but I have done it on this occasion as it demonstrates the grid system. If you look at the diagram below you can see that this uses the Cartesian coordinate system with the origin at the middle of the screen. The x scale goes from -240 (on the left) to 240 (on the right). The y scale goes from -180 (on the bottom) to 180 (on the top).

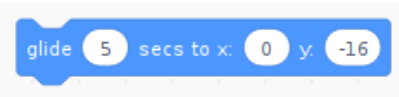


Whenever you need to move a sprite then you can either work it out based on the coordinate system or just move the sprite to the position you want and note it's coordinates.

Moving a sprite – Go to vs. Glide

The quizmaster needs to walk to the centre of the stage. This can be achieved by increasing the x position of the sprite until it reaches 0, but leaving the y position unchanged. If you use the “go to x, y” that you used previously then the character will disappear and reappear in the middle of the stage. As this is a quiz and not a magic show then that's not quite the effect we are looking for.

Instead the glide block will move the sprite slowly over a period of time. The following block will move the sprite into the centre of the screen taking 5 seconds.

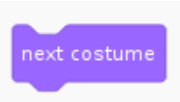


Rather than connect this to the existing block of code just position it by itself in the Code window. Then click it to see what happens.

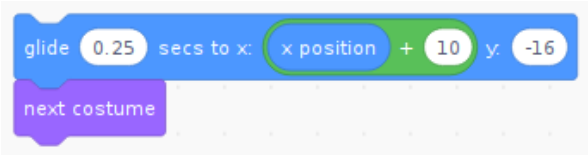
You should see the sprite glide to the centre of the screen. An improvement, but still not very realistic as the legs didn't move. To do that we need to create a loop and change the persons leg positions. This can be done by using the different costumes on the **quizmaster walking** sprite.

If you click on the costume tab you will see 4 different costumes each of which has the foot in a different walking position.

Find the next costume code block and click it (you don't need to copy it into the code area yet). You will see the sprite's legs move. Click again and they move again. If you keep clicking the block at a steady pace then it will appear that the quizmaster is walking on the spot.



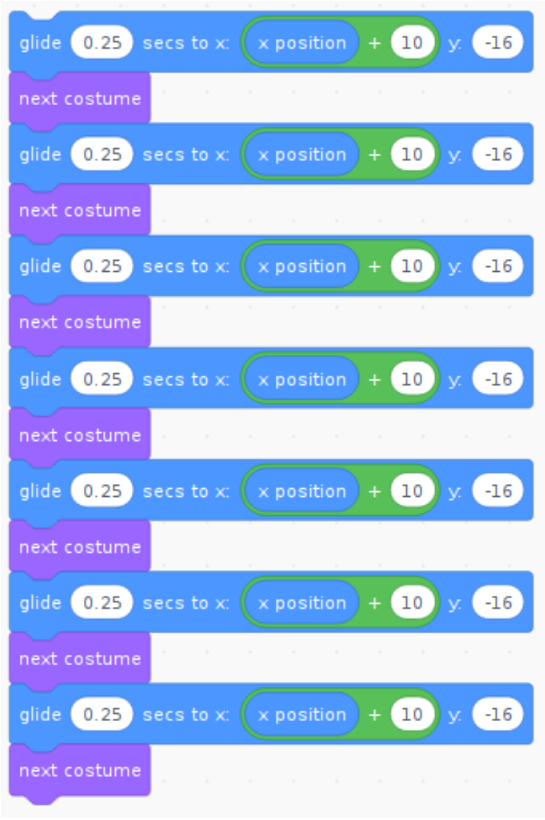
To make them move when walking you need to combine both the next costume block along with the glide block.



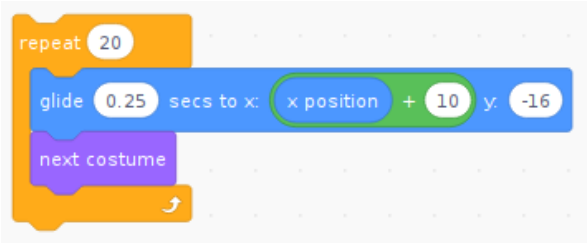
This example uses the relative position of the sprite. So it takes the x position and adds 10 to it. I've left the y field entered manually, but you could change that to y position instead if you preferred.

Repeating code with a loop

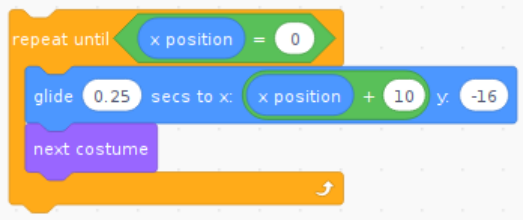
You could now repeat that block by copying it 20 times and you would see the sprite walk into the centre of the screen. It's pretty repetitive to keep repeating the command over again and would create a lot of code. The image below shows only a third of the number of commands needed.



As it is just running the same code over and over again, this can instead be achieved by using a loop. There is a choice of two loops that would work well for this. The first is the repeat loop, which will repeat the action a fixed number of times.



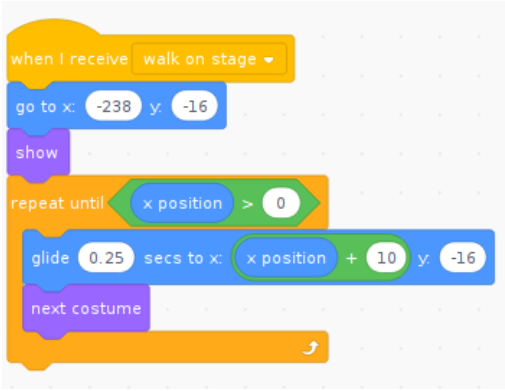
The alternative is a “repeat until” block which can be used until the person reaches the centre of the stage. This is useful if you don't know how many steps are required until the sprite reaches the centre of the stage. The repeat until block is used as below:



There is however a slight problem with this. Give it a try and you may see that the sprite keeps moving beyond the centre of the screen. This is not what we wanted and is known as a bug.

The reason this happens is that the x position is only checked at the start of each loop. As the x position is moved 10 positions in each loop then it will have gone beyond the centre and so never actually equal 0. To fix the bug replace the equals comparison with a greater than sign so that the condition is met whenever the sprites has gone beyond the centre of the screen.

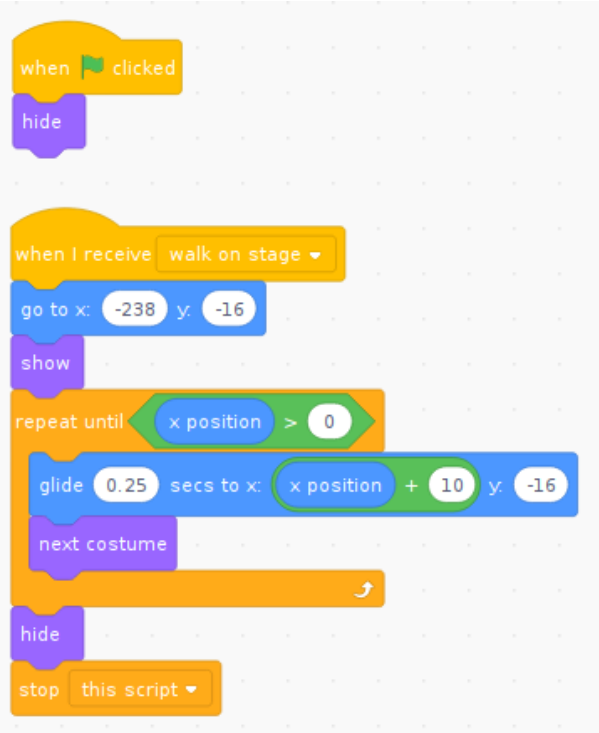
If you fix that bug and then add it to the other code then it should look like the following.



Finally you can then hide the sprite (it will be replaced with the **quizmaster talking** sprite) and then stop that particular script from running.

Complete quizmaster walking code

The entire code on the sprite should then look like the following:



You can test this by running the program with the green flag button. Click on start and the person will walk on to the stage and then hide when they reach the centre of the stage.

The quizmaster talking code

Now that the quizmaster is at the centre of the stage they should change to the **quizmaster talking**. This is triggered by the "start quiz" broadcast message. We could have done this using the same sprite as the one that was walking but with an additional costume. To do that would have meant we would have had to track which costume was selected. So to keep this simple you should use a different sprite called **quizmaster talking**.

Start by hiding the **quizmaster talking** sprite at the start. Click on the **quizmaster talking** sprite and then add the following code.

You may notice that this is exactly the same as the code on the **quizmaster talking** sprite and will run whenever the game starts. This is not really necessary as the sprite will be hidden later, but is useful if the program is ever stopped and then started again.

The next part of the code needs to be called in response to the "start quiz" broadcast message. You also need to ensure that the new sprite is at the same position as the **quizmaster walking** sprite that has just finished walking and become hidden. You can do this using the "go to x,y" block used previously. You can enter the coordinates manually using the x and y coordinates from **quizmaster walking** by looking at it's current values on the sprite properties. Alternatively you can use a built in

feature of Scratch which allows the code to look at the properties of other sprites. The code block for this is in the sensing group, but you may need to change the pull-down options.

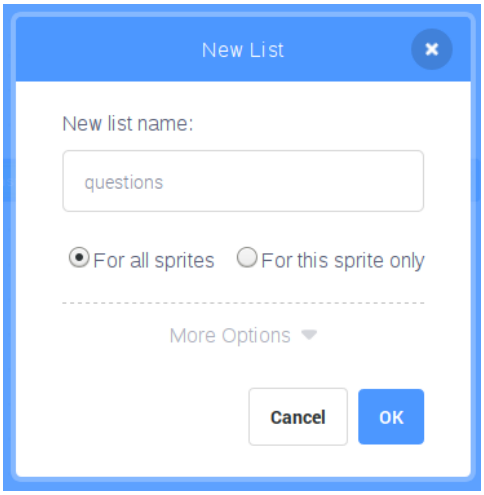
Create the code as shown below

Lists of variables

Before adding the rest of the code to the **quizmaster talking** sprite it will need a list that will hold the questions and answers. Lists are a common feature of programming languages as they allow multiple items to be stored under a single name. Some programming languages call them lists and others call them arrays (sometimes there are technical differences in how they are implemented depending upon the language).

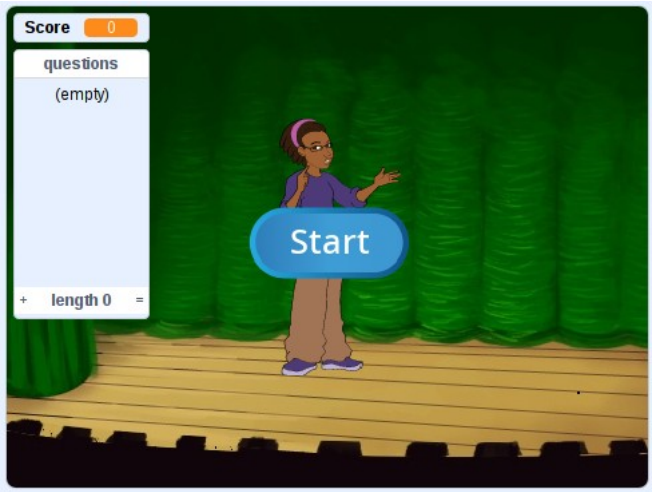
For this particular program the list can be created so that they are visible to all sprites or just within the **quizmaster talking** sprite. I have shown the example of creating it for all sprites.

Click on the **quizmaster talking** sprite, then click on "Make a list" from the Variables group. Enter the name of the list "questions" and select "For all sprites".



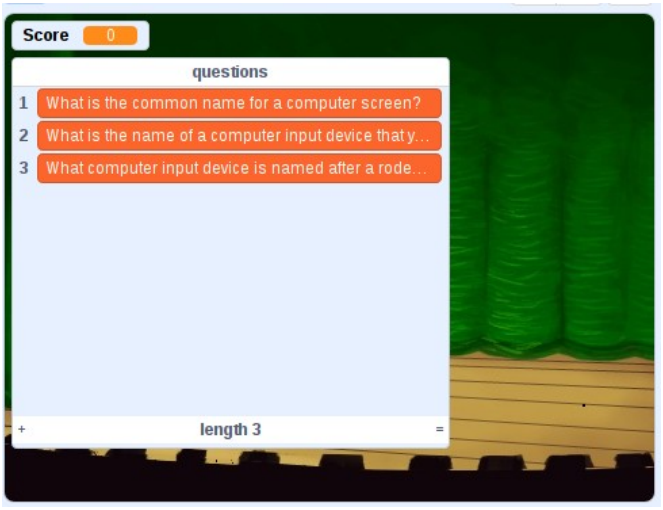
The questions list will be displayed on the stage. The display on the stage can be turn off by removing the tick next to the list, but leave it turned on for now as it makes it easier to add entries. Once there is a list then additional code blocks are available within the Variables group although the colour is slightly darker than the rest of the variables.

We can now add items to the list using the list block that has appeared on the stage.



Click on the + sign at the bottom of the list block to add three questions.
You may find it easier to create all three question entries first (they will be filled with ***) and then change the text afterwards. You can also make the questions box bigger (you can reduce it's size after writing the questions in).

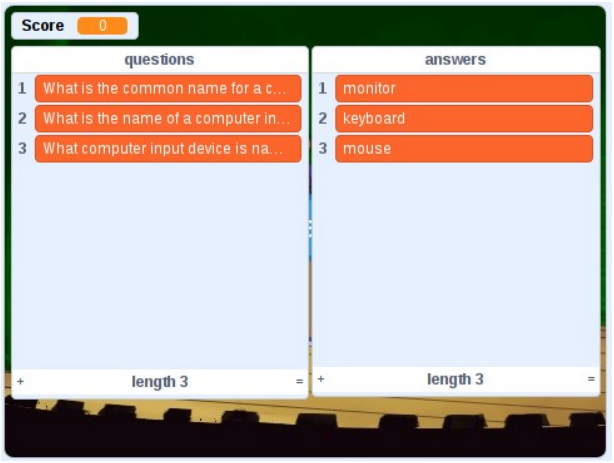
I have chosen three computer related questions. The questions can be on any topic and an appropriate level of difficulty for the age that the quiz is going to be aimed at.



The questions I have added are:

1. What is the common name for a computer screen?
2. What is the name of a computer input device that you can type with?
3. What computer input device is named after a rodent?

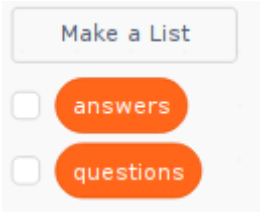
Use the same technique to add a second list called “answers” with the corresponding answers. The position of the answer needs to line up with the same question number (see the numbers on the left).



The answers I have added are:

1. monitor
2. keyboard
3. mouse

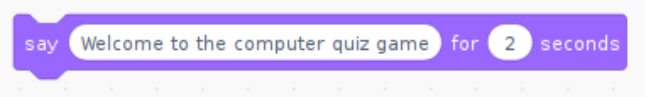
You can now hide these from the stage by removing the ticks from the variable block group.



Asking the questions – say and ask

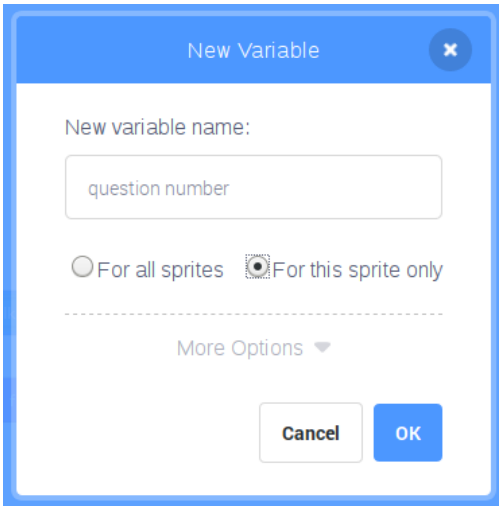
We are now ready to have the quizmaster to introduce themselves and ask the questions.

To have the sprite talk then you can use the say code block. Use the one that has a time next to it as it will give the player time to read the message before moving on to the next block.



You can click on the code block to see what it looks like when the quizmaster talks and if there is enough time to read the message.

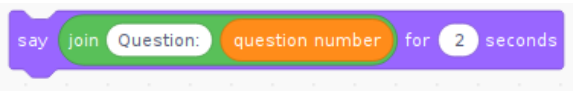
Rather than asking the questions by using a code block for each question you should create another loop which will read out each question one at a time. You will also need a variable called “question number” to keep track of which question is being asked. As none of the other sprite need to know about the “question number” variable you can create it for this sprite only, this reduces the risk of problems if different sprites want to use the same variable name.



Untick the box next to the variable so it doesn't show on the stage. The variable starts at 1, but our question numbers are numbered from 1 upwards, so add a block “set question number to 0” to the current script.

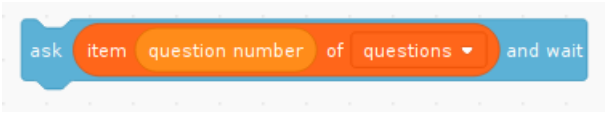
The quizmaster will start by telling the player which question number they are on. Rather than having the sprite say “This is question” then wait before giving the number we can put this on one line by using the join block. This can be inserted in place of the text on the “say for 2 seconds”

block. Don't connect this to the existing script just yet as we will be incorporating this into a loop.



To get a response from the player you can use the “ask and wait” block to to get a response. This code will ask a question and store the players response into a (pre-defined) variable called answer. Note that his is singular answer, rather than answers which is the name of a list you created earlier.

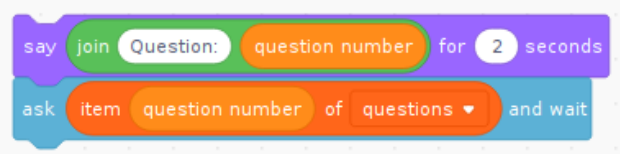
As our question is stored in a list we use the “item 1 of questions” block which provides the contents of the list at a specified position. Use the variable “question number” to determine which of the questions to show.



The ask block will display the text from the questions list on the screen and then provide an input field that the player can type their response into.

Note that lists in Scratch start from position 1. This is how Scratch works and is more natural for teaching to children, but most other programming languages start at 0 which is how computers start counting.

We now have the following that will ask the player the question and wait for them to reply. The response will be stored as answer which is a block available from the Sensing category (rather than the normal variables).

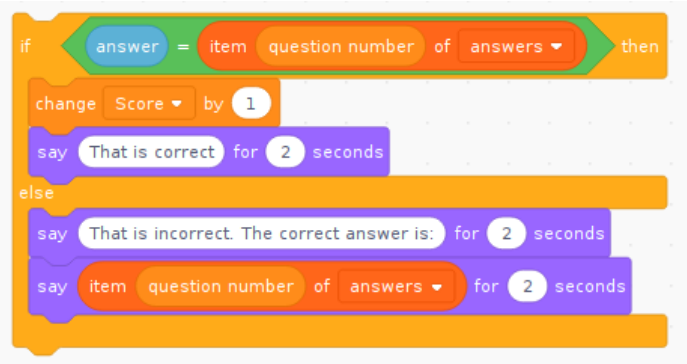


Checking for a correct answer – if, else

The answer provided by the player is stored in the answer variable which we can comparing against the expected answer using an if code block. The if code block will only run a block of text “if” the condition is true.

To give a different message for an incorrect answer you can use the “if , else” block, which will run different code depending upon whether the condition is met or not.

The code for the check is as below:

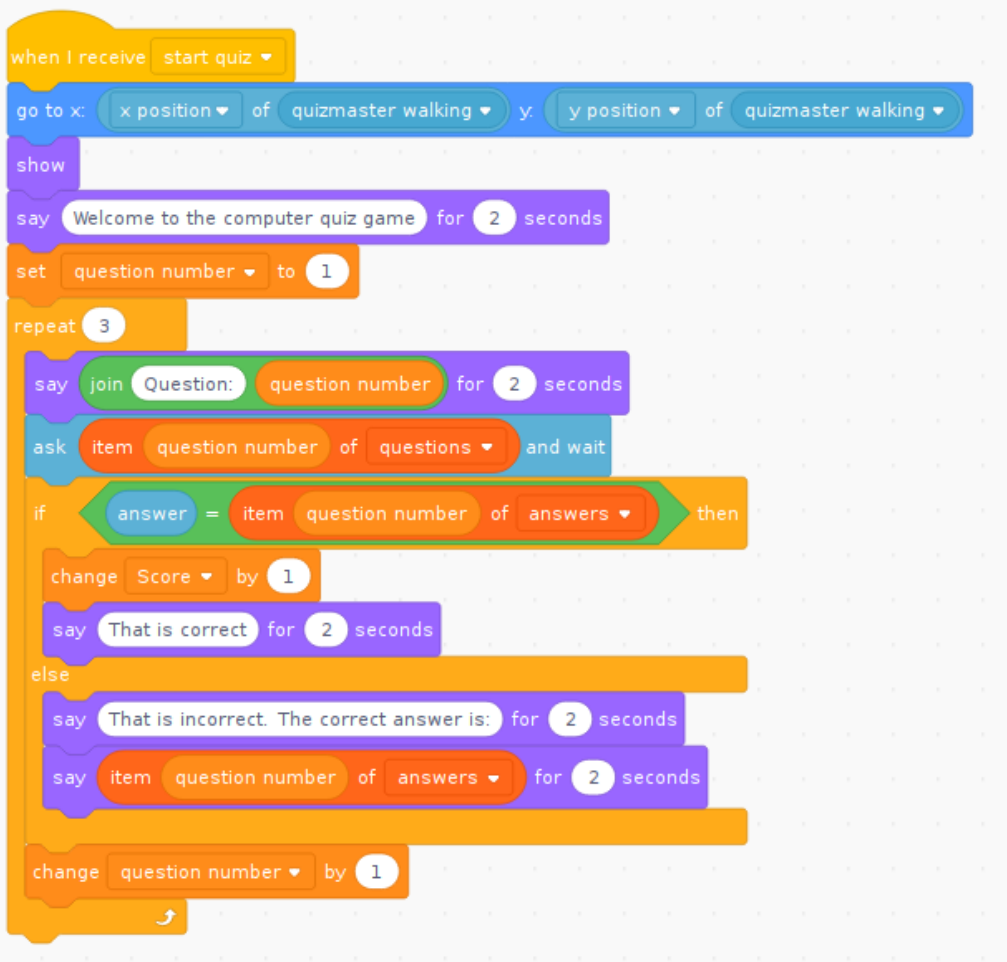


If the answer given by the player “answer” matches the answer stored in the answers list then the first section of the code is run which increases the score by one and says “That is correct”. If the answer does not match the expected answer then the else clause is run which says that the answer is incorrect and gives the correct answer.

Read through the code above and make sure you understand what is happening, we have used similar blocks before if you want to check anything. Normally you can just click to test a block of code, but this is dependant upon the other code having run first so you will need to wait until it is incorporated with the rest of the code before testing.

In Scratch the answer will only be considered as correct if it is exactly the same as the value in the list. Unlike other programming languages comparing strings in Scratch is not case sensitive so the following will all be considered the same: monitor; Monitor and MonITor. Any other differences (including a space character at the end) would not be considered the same answer.

This block of code needs to be run three times, once for each question, so it can be included in a repeat loop. When you join the blocks together you should have a long script as follows:



I've also increased the question_number at the bottom of the repeat loop so that we move on to the next question.

Telling the player how they did

We can now provide additional output back to the user telling them how well they have done. In this case we give them a message telling them if they "Got full marks", "Did reasonably well" (between one and two correct) or if they need to "try again" (got zero marks).

If there were only two possible outcomes then we could use a single if-else block as we have used previously. When we have more than two then we need to nest two if-else blocks. So if the first condition is met then we give the full marks message, but if not (else) we have to perform a second test to see whether they got some or no marks.

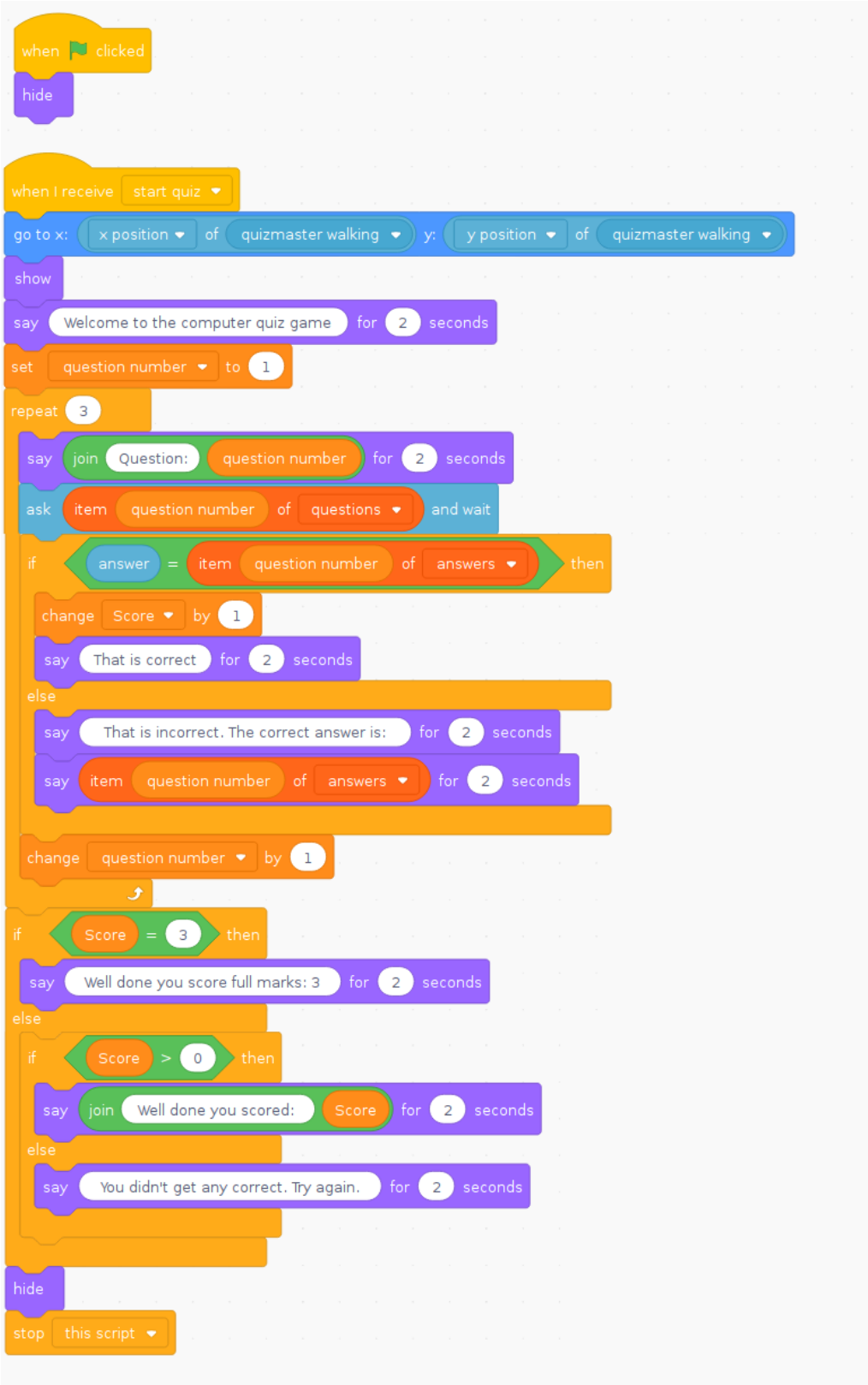
I have used an equals test for the first condition and then a greater than test for the second. We could have used an equals test for the condition where no questions were answered correctly, or we could have used a not equals to zero for the one where one or two answers were correct. The method used doesn't matter as long as the logic matches up with the test to be performed.

I have used a join so that it can give the actual score in the response to the player.

You can then hide the quizmaster talking sprite and stop that script.

Completed quizmaster talking script

The completed code is shown below:


when clicked
hide
when I receive start quiz
go to x: x position of quizmaster walking y: y position of quizmaster walking
show
say Welcome to the computer quiz game for 2 seconds
set question number to 1
repeat 3
say join Question: question number for 2 seconds
ask item question number of questions and wait
if answer = item question number of answers then
change Score by 1
say That is correct for 2 seconds
else
say That is incorrect. The correct answer is: for 2 seconds
say item question number of answers for 2 seconds
change question number by 1
if Score = 3 then
say Well done you score full marks: 3 for 2 seconds
else
if Score > 0 then
say join Well done you scored: Score for 2 seconds
else
say You didn't get any correct. Try again. for 2 seconds
hide
stop this script

Testing the game

The game is complete and now it's time to start testing. Click the green flag icon to start the programming running and then click start to start the quiz. Type in your response to each of the questions and see if it works.

When testing it's important to check that it works if the answers are incorrect as well as if the answers are incorrect. Also what happens if the user does something that they should not be doing. For example what if the user submits and empty response (just clicks on the tick to submit) or uses some invalid characters? Sometimes programs can crash when presented with a character that it doesn't expect such as a closing quote ” part way through a string. Fortunately Scratch is quite good at handling those kind of things, but other languages may not be so good.

In an ideal world you would test every different possible combination of correct vs incorrect answers. You could test with all answers incorrect; then with the first answer correct, followed by two incorrect answers; then with the second answer correct, but all others incorrect etc. To test all the different combinations can be difficult or even impossible to achieve for some programs. With our simple program with only 3 questions just testing a full set of correct vs incorrect answers will provide 2^3 different combinations which is eight individual tests. If you were to have a quiz with 10 questions then that would jump to 2^{10} which is 1024 different combinations of correct and incorrect answers. If it's not possible to test every different possibility at a minimum you should test the different outcomes. In this case the result when answering all the questions correct, the result when getting some answers correct and the result of answering all questions incorrectly. Then test some different combinations of correct and incorrect answers and see if it gives the correct answer for each test.

Summary so far

It's taken us a while to get here, but we do now have a working game that can be played. If you are logged in to a Scratch account you can click on Share to make the game available for others to play.

This game is available from:
www.penguintutor.com/programming/scratch

By creating the quiz game through each sprite it should be possible to see how it works. We have covered many of the techniques used to make games in Scratch including: starting code, loops, conditions, variables, lists and user input. This should be enough to get started on creating your own programs in Scratch.

This guide has also included information on how to test your program.

Next steps

Whilst this quiz is fully working there are still some areas where it could be improved. Here are a few ideas that you could try:

- Have the quizmaster walk back off the stage at the end
- Have appropriate music playing as the quiz master walks on and/off the stage and when the results are announced.
- Add more questions, or questions on a different theme.

Further resources

There are a number of free resources for learning Scratch and programming in general. I have provide a number of useful links below.

- www.penguintutor.com/programming/scratch
  Accompanying website for this guide.
- scratch.mit.edu
  Scratch website
- www.themagpi.com
  Free online magazine for the Raspberry Pi (includes Scratch examples)
- www.codeclub.org.uk
  After school club using Scratch

Some of the further resources will be based on Scratch 2. The code examples should still work the same, but the Scratch 3 programming environment is different and some of the options will look a little different.

Copyright