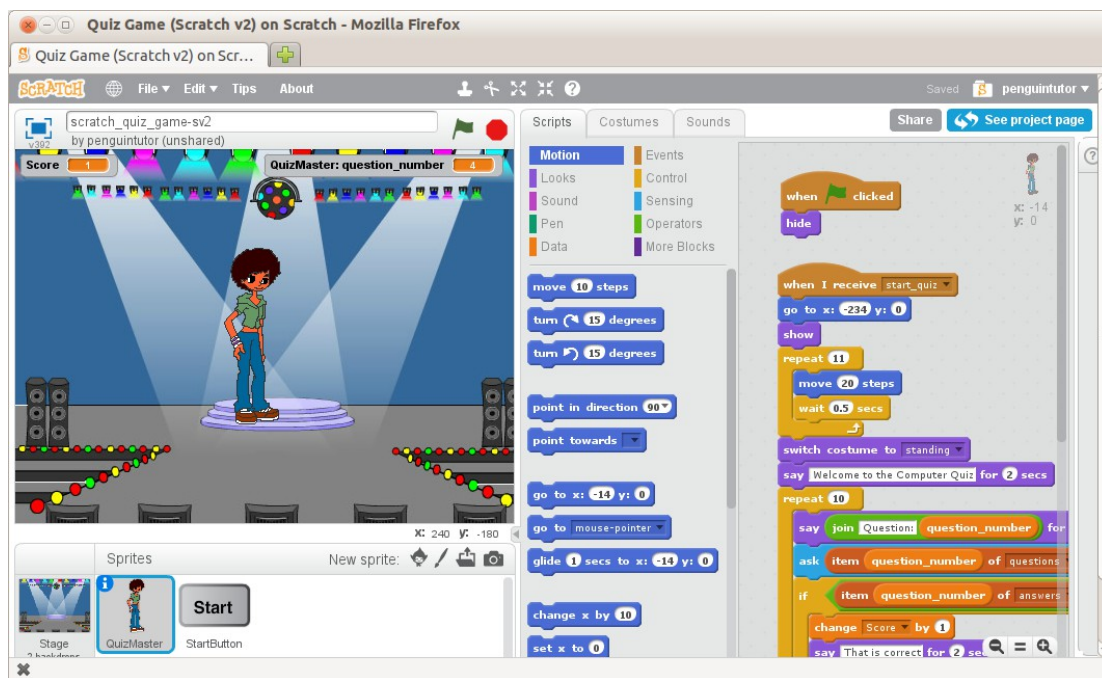


Scratch Programming for Primary School Teachers

Quiz Game



For Scratch Version 2

Stewart Watkiss



PenguinTutor.com
www.penguintutor.com/teachers

January 2014

About this guide

During my preparation for an introduction to Scratch training session for a UK primary school, I wanted a guide that the teachers could use to create a practical program in Scratch. I wanted that guide to cover the key aspects of computer programming in the national curriculum. I was able to find many guides that were aimed at the school children directly but it would take a lot of time to work through from the beginner level in Scratch to cover the different programming techniques in the curriculum. Conscious of the work load that teachers are under, especially with the introduction of the new curriculum I wanted a guide that could see them create a working program in around an hour that would cover all those aspects. Unable to find a suitable existing guide I created this quiz game as a method of introducing the different techniques.

Working through the guide will create a computer game and in doing so introduce all the programming techniques from the primary school teacher training session. This aims to cover the main programming techniques and knowledge required for the Key Stage 1 and 2 curriculum. 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.

The guide is not intended for teaching direct to primary school children as it introduces a lot of techniques in a single game. It is anticipated that this is instead used by a teacher wanting to understand the different techniques prior to them running a more step-by-step course within their classroom. It may also be useful for older children who have already had an introduction to Scratch, but have not yet covered all the different programming techniques used.



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

About the author

Stewart Watkiss graduated from The University of Hull, UK, with a masters degree in electronic engineering. Since then he has been working in the IT industry for the last 17 years, including Linux and networking.

Linux has been his passion for many years and he has gained certification at LPIC-2. He runs the website PenguinTutor.com providing information on learning Linux, electronics and the Raspberry Pi. After working on several projects with the Raspberry Pi both on his own or with his children he signed up as a STEM ambassador to pass some of his knowledge on to school children.

He has also been technical reviewer of the book “Learning Raspberry Pi with Linux”, published by Apress, and a volunteer guest writer for The MagPi Raspberry Pi magazine.

Table of Contents

Introduction.....	3
About Scratch.....	3
Getting started with Scratch.....	3
A. The Scratch stage.....	5
B. The sprite list area.....	5
C. The sprite edit area.....	5
D. The code blocks area.....	6
Designing a game.....	7
Design Specification – Quiz Game.....	8
Aim.....	8
Sprites.....	8
Variables.....	8
Running Scratch.....	9
Creating the code.....	9
Setting up the Sprites.....	9
The main program script.....	13
The start button.....	15
Creating the list variables.....	16
The Quiz Master Script.....	19
Testing whilst creating the program.....	25
Debugging.....	25
Test 1.....	26
Test 2.....	26
Test 3.....	26
Test 4.....	26
Other debugging techniques.....	28
Telling the player how they did.....	28
Leave the stage.....	29
The completed script for the Quiz Master sprite.....	31
Summary so far.....	32
Next steps.....	32
Further resources.....	32
Copyright.....	33

Introduction

Following this guide will create a computer quiz game using the Scratch programming languages. It is recommended that this is worked through in Scratch ending 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/teachers/. 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 available for free and can be run on various different computer platforms including a web-based version.

Scratch is usually considered the first choice when looking at a PC based programming language for teaching primary school children. It can be installed on most school computers and is available pre-installed on the Raspberry Pi education computer when running the Raspbian operating system. 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 2 which is primarily web-based and accessed through a web browser using the Flash plug-in. This document is based on Scratch version 2. For Scratch version 1.4 (as installed in many primary schools and on the Raspberry Pi) see the alternate version at: www.penguintutor.com/teachers

Scratch website: scratch.mit.edu/

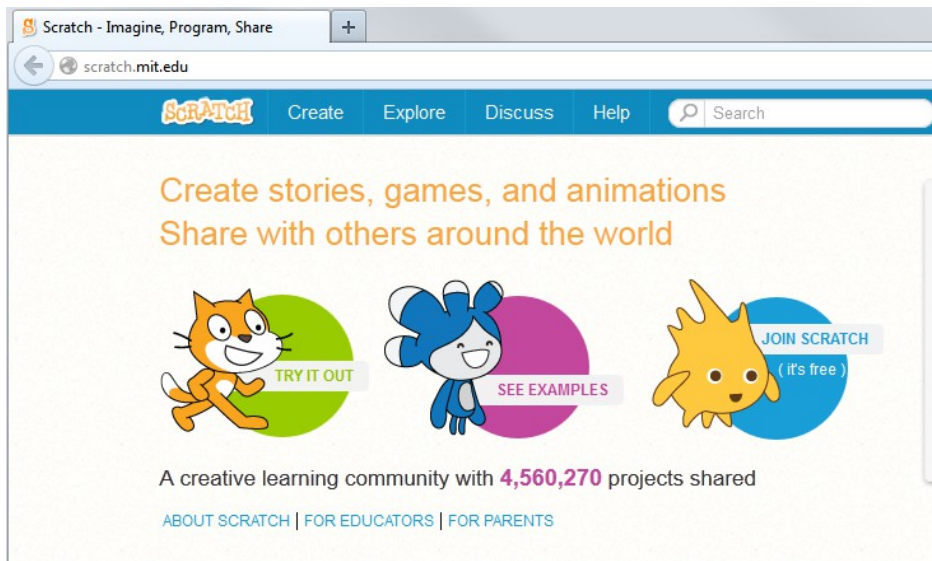
Unfortunately any games you create with Scratch version 2 cannot be opened using Scratch 1.4. If you have some computers that don't run Scratch version 2 then you may prefer to use Scratch 1.4.

Getting started with Scratch

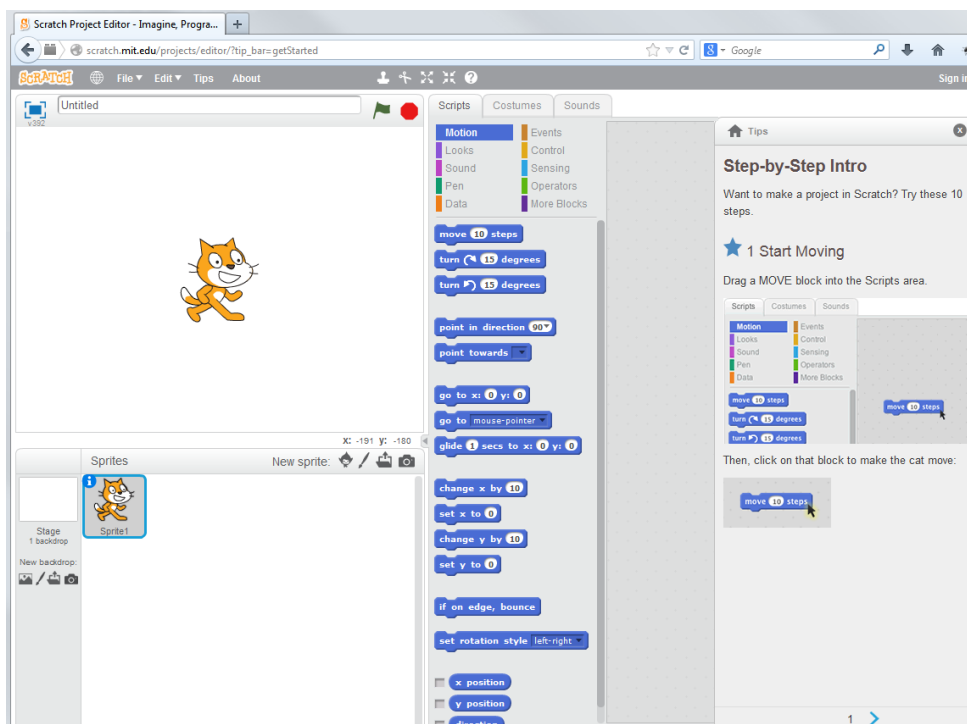
Scratch uses a drag-and-drop approach to programming making it very easy to use. It comes with a development environment that can be used to create programs without needing to learn any complicated rules about the computer language.

To run Scratch version 2 you will need a recent web-browser with the Flash plug-in.

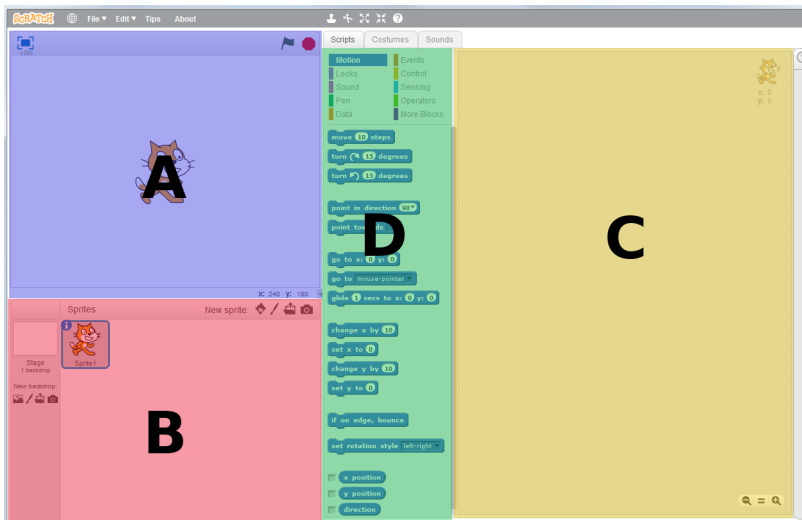
The initial web-page is as shown below:



Click on the Create option in the top of the screen to start 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.

There is also 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 lights used to create the show. In Scratch a sprite can have it's own costumes, scripts 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 left part of the sprites list area.

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 Scripts, Costumes and Sounds.

In Scratch a chunk of program code is called a script and is edited on the scripts tab. It is fairly common to have several scripts associated with each sprite. The stage can also have its own scripts 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 background 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 suggested as 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 scripts. 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 Scripts tab is selected.

The blocks of code are grouped into eight different and are colour coded along with the code groups.



The groups are:

Motion – moving a sprite around the screen.

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

Sound – make noises and play music

Pen – used for drawing

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

Events – blocks that indicate when a script should start running

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

Sensing – detect the position of the sprite and the mouse

Operators – perform logical and mathematical operations

More blocks – custom custom blocks

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 scripts 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

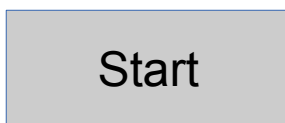
Aim

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

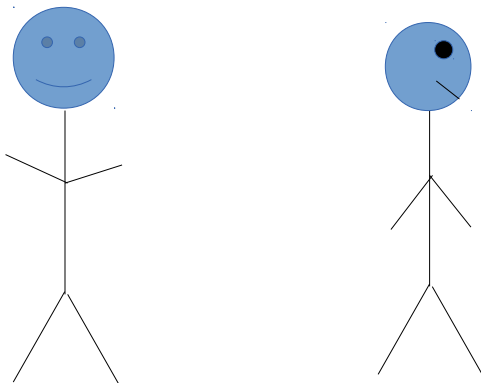
A list of sprites to be used in the game.

StartButton - “Click to start”



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

These are stick drawings, will use existing sprite costumes 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

Running Scratch

There are two versions of Scratch, one of which runs in a web client and the other is installed onto the local computer. These instructions are based on Scratch 2 which is web-based.

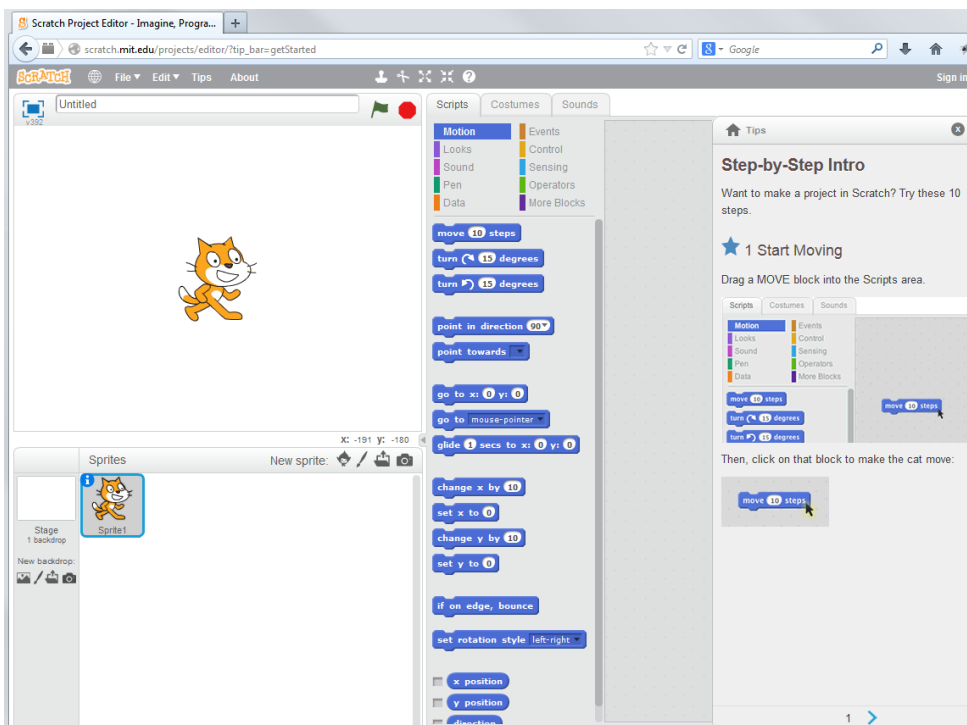
It will only work with a recent browser with the Flash plug-in installed.

Creating the code

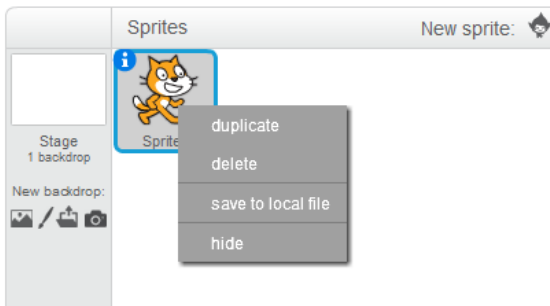
The order of creating the game comes down to personal preference. For this game I have created the sprites first which means that we can set-up the visual aspects (sprites and costumes) first and then concentrate on the code afterwards.

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.



We won't be using the cat so delete Sprite1 by right clicking on the Sprite in the palette and clicking delete.

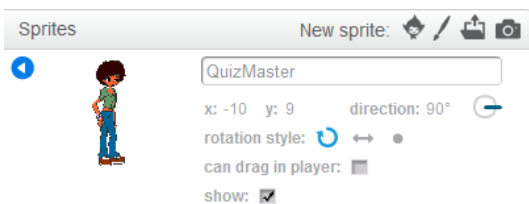


Now we can add an image of a person who will be our quiz master. Create a new sprite from library (see icon shown to the right) and find an appropriate person facing forwards. I've used Girl6. I have chosen this image as the sprite has two costumes, one of a girl facing forward and the second of the same girl walking.



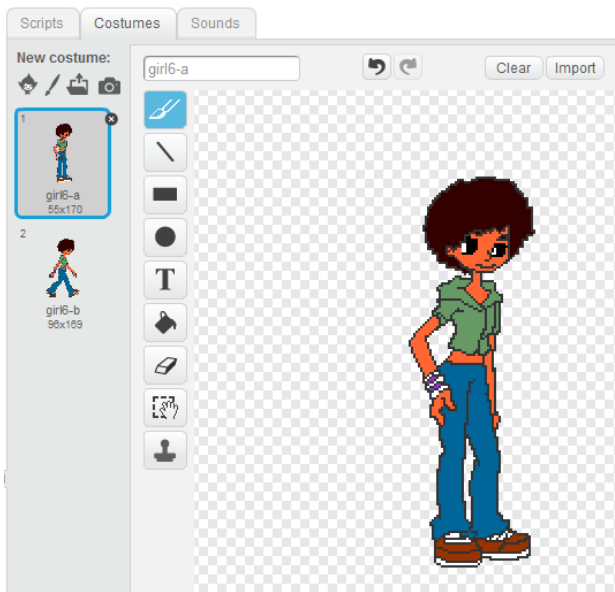
When using the new sprite button then you only see the default costume for a particular sprite. When accessing the sprite library through the costume selection (see later) then you will see all the costumes available.

When the new sprite is created it will be named after the name in the library. We can rename the sprite by clicking on the blue i icon and changing the name to QuizMaster. 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.



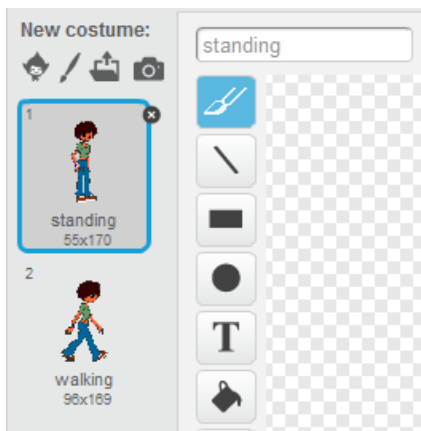
To add a bit of an introduction we will have the quiz master walk onto the stage. For this example we will have the person look as though they are walking.

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.



In this case we have a costume with the girl looking forward and one with the girl walking. The icons above the costume list can be used to add a new costume if required.

The different costumes can be renamed using the text box. I have renamed these standing and walking which will make the code easier to read later.



A future improvement would be to have additional images showing the right and left legs moving forwards and backwards. We have not included that here, but it is included as a suggestion once the program is complete.

We will add the code to go with the QuizMaster sprite later. Next we can add the start button sprite.

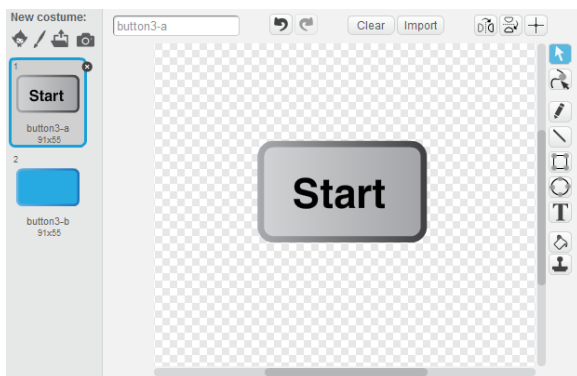
Use the “Create a sprite from a file” option and this time choose a button from the Things category. I have used Button3.

Initially the button is plain with no text. We will need to edit the costume to manually add the text.

Rename the sprite to StartButton and click on the costumes tab.

Using the 'T' Text button type “Start”. The text will be created, but it will need to be moved to the centre of the button image, which can be done using the arrow button.

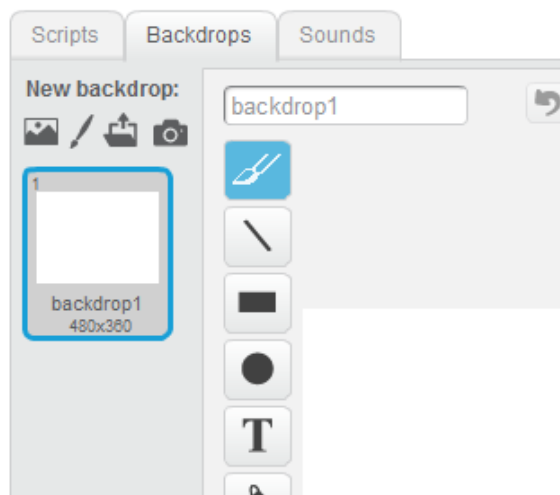
The button should now look similar to below:



We can also set the Stage background image to an appropriate scene.

Click on the Stage which is next to the sprite list.

The stage will load into the edit area, but instead of having a Costumes tab instead there is a Backdrops tab.



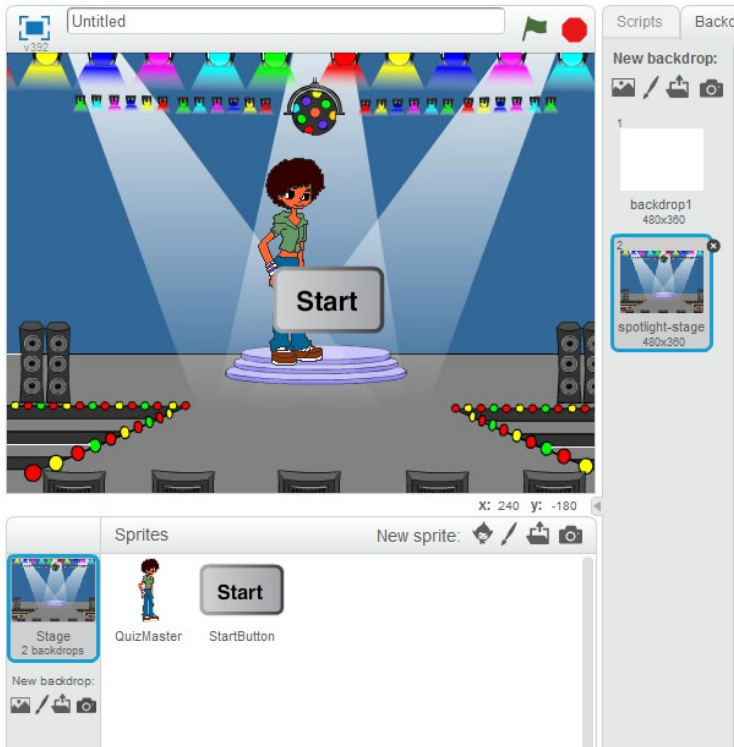
Click on the New backdrop from library button and select an appropriate backdrop.



I have chosen Indoors and spotlight-stage.

You should now have a Stage, QuizMaster and StartButton.


The QuizMaster and StartButton sprites are overlapping, but we will be only showing one at a time when we complete the program.



The main program script

For this example we will have a master script that will send and receive messages from the other sprites. This script will be within the stage.

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

Click on the stage Scripts tab and add the “When  (green flag) clicked” block to the scripts area by dragging it from the Events area. 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.

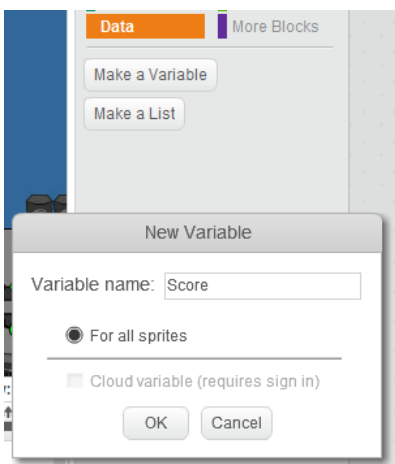


The forever loop will run the whole time that the program is running and anything that is included in that will be continually repeated. How often a loop is repeated depends upon the different steps that are called within the loop. In this case the forever loop will launch most of the other scripts and 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 be perfectly valid too, but the forever loop is used here to show how a program can keep running after it would otherwise end.

Now create a new variable by clicking on the Data category and then “Make a Variable” and call it “Score”. As we are creating the variable whilst on the stage edit screen the only option is “For all sprites” which means that all the sprites can access the variable. We will see additional options when we create a variable for a sprite later.



When we have created a variable there will be additional blocks added for reading and changing the variable. Use the “set Score to 0” and put that 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.

We want to be able 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 StartButton 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_startbutton”; This is done using the small down-arrow and choose “new message...”.

We will also need to be able to communicate with the QuizMaster script and so we can create an

additional broadcast that the QuizMaster script can listen for. So add a second broadcast and wait block with the message “start_quiz”.

The script should now look as below:



That is the first script complete, although it won't actually do anything until we create some code to respond to the broadcast messages.

Before we 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 we go into 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.

We set the Score variable to 0 – so far the player hasn't scored any point.

We then send a broadcast to tell the StartButton sprite to show itself and we then wait until it the StartButton script finishes. The script on the StartButton will run until the button has been clicked by the user, but we haven't created that script yet.

When the StartButton script finishes then we will continue to the next broadcast which is going to tell the QuizMaster script that it's time for the quiz to start.

When the script on the QuizMaster finishes (after all the questions have been answered) then we return to the forever loop. As we have reached the bottom we 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, as it's name suggests the broadcast actually goes to all the sprites including the stage. It's just that in our case only one of the sprites will actual act on each of the broadcast messages. 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 our script will hide the button and allow the program to start.

The StartButton 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 we need to do is to show the button. First add the block “When I receive show_startbutton” from the control category. Then add a show block from the Looks category. Ensure they snap together.

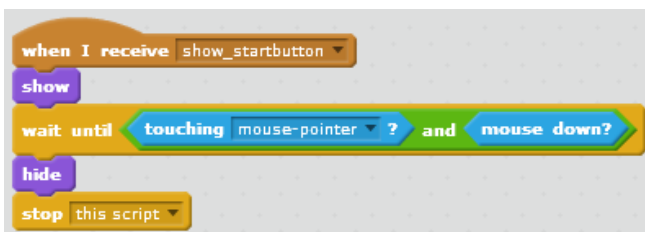
We now want to wait until the mouse is clicked on the button. This is done using the “wait until” block. As its name suggests the script will wait until the specified condition is met.

To be able to check for both the mouse being over the button **and** the mouse button being clicked we 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, we then also need the mouse button to be pressed before the and condition is met. It should now look as show in the following code:



When the button is clicked we then need to hide the button and stop this script so that the main script can continue.

The finished script should look as follows:



You can test the scripts 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.

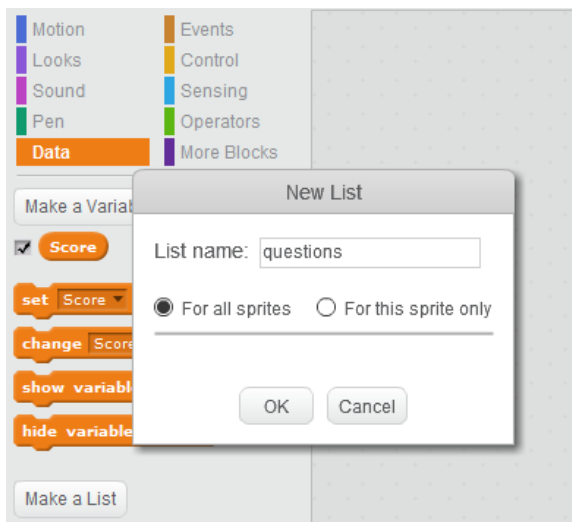


The “stop script” block is not actually required. When the code reaches the end of a script then it will stop that script automatically. It is however good practice to show the end of a particular script and helps to make it easier to understand. In this game I have added the stop script on the end of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

Creating the list variables

Before we add the scripts to the QuizMaster sprite we will add the list variables that will hold the questions and answers. For this particular program these can be created so that they are visible to all sprites or just within the QuizMaster sprite. We will create the variable so that it can be seen by all sprites as that will make it simpler if we wanted to be able to add additional questions in future.

Click on the QuizMaster sprite, then click on “Make a list” from the Data blocks, enter the name of the list “questions” and select “For all sprites”.



As well as showing the questions list as an additional block within the variables another nine blocks of code appear under the list section. These blocks are used to read or change the values of the list. We will use some of these later so that the code can display the questions, but for now just ensure that there is a tick next to questions so that we can see the list on the stage.



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

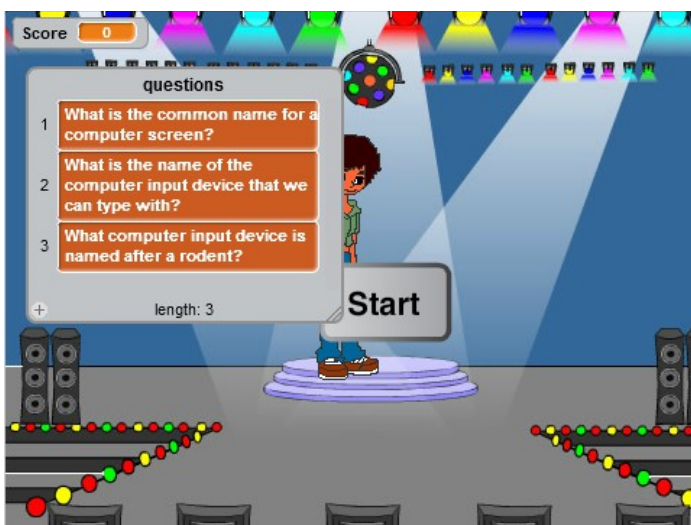


Click on the + sign at the bottom of the list block to add three questions.

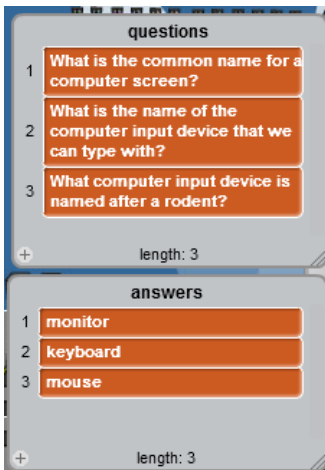
You may find it easier to create all three question entries first (they will be filled with ***) and then change the text afterwards. You should also make the questions box bigger (you can reduce it's size after writing the questions in).



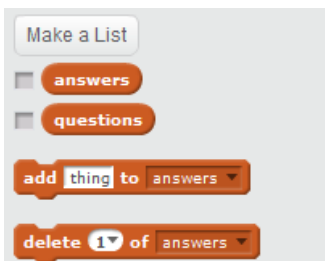
I have chosen three computer related questions. The questions can be on any topic and an appropriate level of difficulty for the age that the quiz is going to be aimed at.



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).



We can now hide these from the stage by removing the ticks from the variable block section, or right click on the variables on the stage and choose hide.



The Quiz Master Script

We are now ready for the scripts for the QuizMaster sprite.

First it is a good idea to hide the Start button from the stage. This is not strictly necessary, but will make it easier to see how the character will look without a start button in front of them. This can be achieved by opening the StartButton sprite and then double clicking on the hide block in the looks section (there is no need to actually drag the hide block to the scripts area, you can still activate that block from the palette area).

We will start with the QuizMaster sprite hidden until the start button has clicked.

Click onto the QuizMaster sprite and add the following script in the scripts area:



Note that this is the second “when green flag clicked” script we have created (the previous on the Stage); both this and the one on the stage will start running when the start flag is pressed.

We will now add a second script to the QuizMaster sprite. This one will not run initially, but will wait until after the start button has been clicked. We have already set up a broadcast from the stage

script, which will broadcast a message after the start button script completes, so we will listen for that.

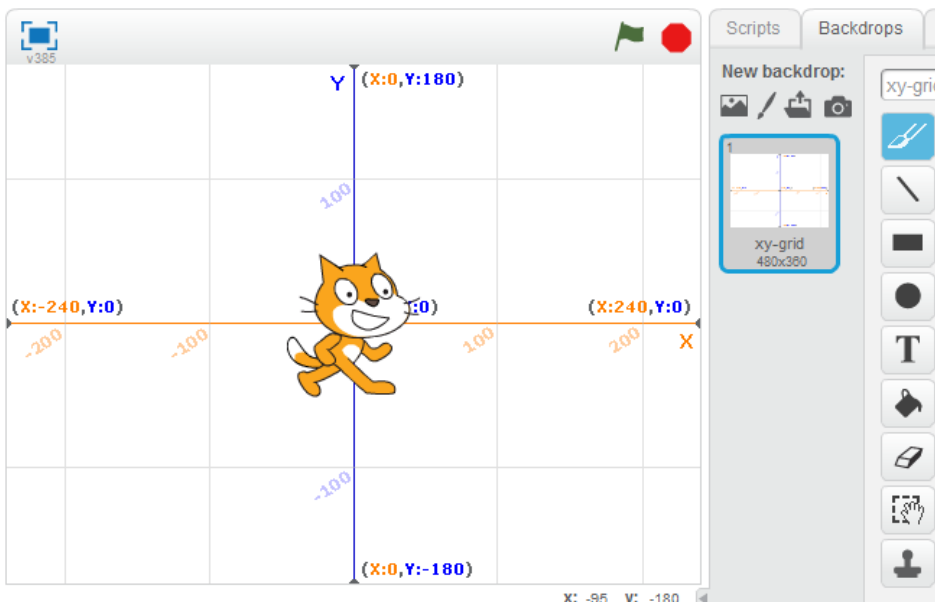
Add when I receive start_quiz (use the pull-down message to change the broadcast to listen for).



We want the quiz master to walk in from the left. Scratch uses an x-y co-ordinate system, so we can use that to tell Scratch where to move the sprite to. The X-axis goes from -240 to +240 and the Y-axis from -180 to +180, the centre of the screen is with both values at zero (0,0).



You can get an image showing the x-y coordinates by importing the xy-grid backdrop image into a scratch program.



An easy way to get work out a specific position is to place your mouse to the required position and then read the X and Y co-ordinates shown in the bottom right of the stage. Alternatively move the sprite to the correct position and the code blocks in the motion category will be updated to the current position.



We can now add a “go to x: y:” code block to move the sprite to the appropriate position. I have used x as -234 and y as 0, which positions the sprite at the left hand side of the screen.

Note that whilst we have moved the sprite, when the program starts the sprite is hidden, so now it is at the correct position we can set it to show.

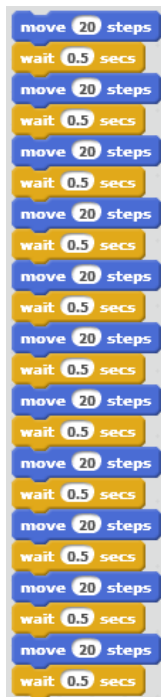


We now want the sprite to move as though it's walking. It's possible to get a realistic representation of walking by having multiple costumes with the legs at different positions. The costume we chose does not have the corresponding costumes and for this example it is sufficient to have the person move across the stage a bit at a time.

If your sprite is still showing the standing position change it to the walking position (this can be done within the costumes tab, do not add anything to the script yet).

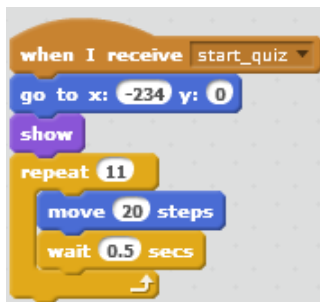
We shall have the character move a short distance at a time followed by a pause and then move again until they reach the correct position.

This could be achieved using:



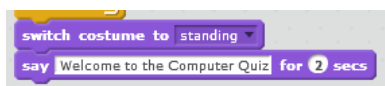
But that is an awful lot of commands. If we had to create an entire program by listing every single step then it would get very long indeed. It would also be difficult to change the distance that the person moves. Instead we can replace this with a single loop repeated 11 times.

So we can replace the previous block of code with the following:



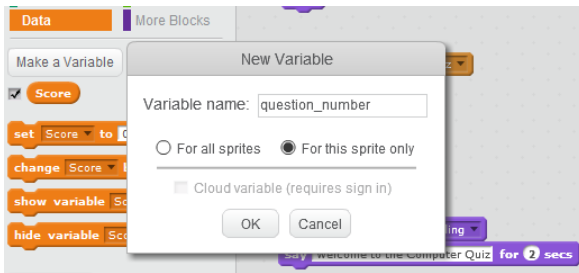
which uses much less space.

We can then swap costume to the standing costume so that the quiz master faces the front of the stage, and say a welcome message.



Rather than asking the questions by using a code block for each question we are going to create another loop that will read out each question one at a time. We will also need a variable called `question_number` to keep track of which question we are reading out. As none of the other sprite need to know about this `question_number` variable we can create it for this sprite only, this reduces

the risk of problems if different sprites refer to the same variable name.



We will start with the quiz master 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.



We can then use the “ask and wait” block to get a response, which will be stored into an (pre-defined) variable called answer.

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.

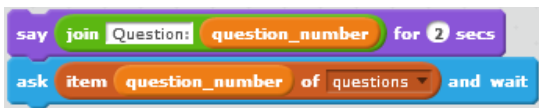


The ask block will create the text on the screen and then provide an input field that the player can type their response into.



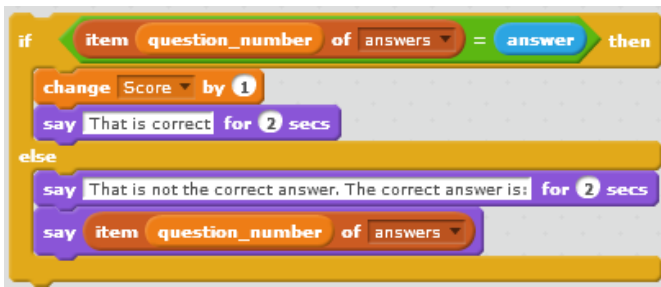
Note that lists in Scratch start from position 1. This is how Scratch works and is more natural for teaching to children, but most other programming languages start at 0 which is how computers start counting.

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.



The answer provided by the player is stored in the answer variable which we can comparing against the expected answer using an if block. We want to give a different message for an incorrect answer so we use the “if – else” block.

The code for the check is as below:



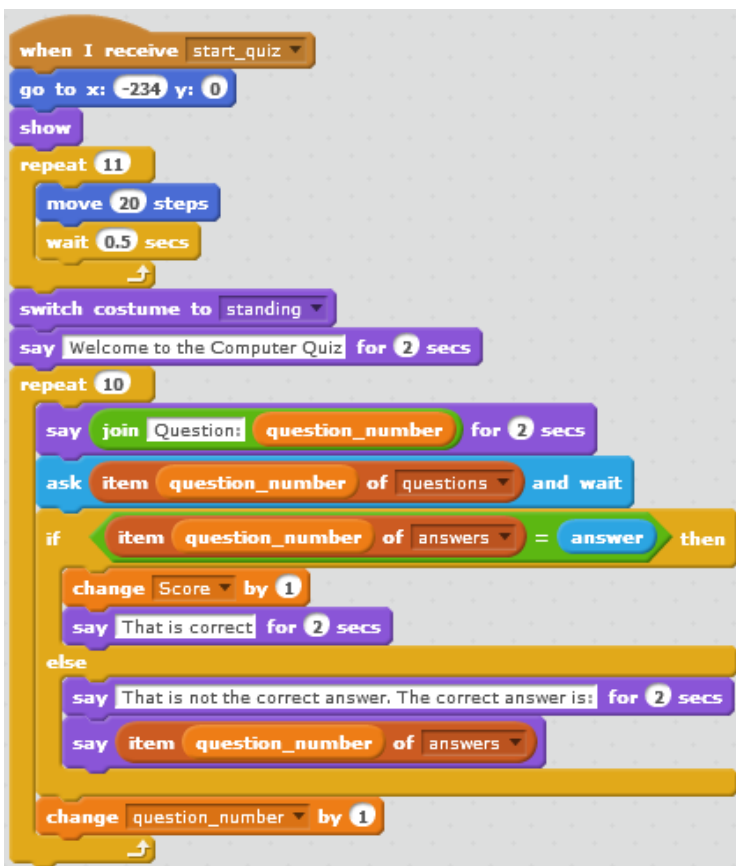
If the answer given by the player `answer` matches the answer stored in the answers list then the first section is run, otherwise the section after the else.

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 double-click to test a block of code, but this is dependant upon the other code so you will need to test the rest of the code together.



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.

We want to run through this three times, once for each question, so we can incorporate this into a repeat loop. When you join the blocks together we 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.

Testing whilst creating the program

We are still not finished yet, but it's a good point to test the code and see if it is behaving as we expect. This will make fixing any bugs easier than if we wait until we have completed everything. First check that the code is entered the same as above. It can be easy to make a mistake on such a large block of code.



In professional software the code will need to be thoroughly tested as it is created. In some cases there may be an entire program written that can perform automatic testing on the code so as to try and identify bugs that would otherwise remain hidden and cause problems in certain circumstances.

To make it easier to see what is happening we will set the `Score` variable to display on the stage (tick box next to the variable) so that we can see the score increasing. The program doesn't yet give any feedback on the current score so we can check this value is changing as we expect.

Click on the green start flag and then the start button and see what happens.

There are a couple of things we haven't implemented yet, so don't worry if the quiz master looks like they are standing when they should be walking and that the quiz master does not get hidden at the end.

- Is the code working as it should?
- Does the score increase correctly?
- If anything isn't behaving as you expect can you think what the problem is?

Debugging

If you haven't guessed yet the program isn't running quite as it should. When writing a program then it is common for there to be errors in the code, in computer terms we call these bugs and the process of finding and fixing the bugs debugging. Bugs can exist because of a variety of different reasons, some common mistakes are

- Putting a block of code inside the wrong section (such as after a loop rather than inside the loop).
- Referring to the wrong variable (for example referring to questions when you should be referring to answers).
- Choosing the wrong command (changing a value by adding to the existing variable when you mean to setting the variable to a new value – or vice-versa).
- Typing errors (not so much in Scratch where the blocks are built, but easy to make in other programming languages).

First we need to have a way of testing the code so that we can find the bugs. You can just try randomly getting some answers right and some wrong, but that could miss common bugs. To test

for bugs we will create a set of tests that covers a number of possible situations.



It is not always necessary to test the behaviour as formally as we will do here, but I have created some formal tests as these will help explain the logical reasoning required when debugging problems. Here are a few tests that we can do that will help find some bugs, they include the input we should type and some of the expected outputs from the program.

Test 1

Answer all questions correctly.

Expected score: 3

Output from sprite: After each question the quiz master should say “correct”

Test 2

Answer first question incorrectly, the rest correctly.

Expected score: 2

Output from sprite: After first question the quiz master should say “incorrect” and provide the correct answer, after subsequent questions the quiz master will say “correct”

Test 3

Answer first two questions correctly, the third incorrectly

Expected score: 2

Output from sprite: After first two questions the quiz master will say “correct”, after third they will say “incorrect” and provide the correct answer.

Test 4

Answer all questions incorrectly

Expected score: 0

Output from sprite: After each question the quiz master will say that the answer was “incorrect” and provide the correct answer.

So let's run the test and see what happens. Click on the green arrow and then enter an appropriate response for each of the tests.

Test 1 is successful. The score is displayed correctly as 3, but does go back to 0 when it starts the loop for the next attempt, which is fine.

Test 2 fails. The score is updated correctly to 2 (returns to 0 on the start of the next loop), but it does not display the correct answer after the first incorrect answer.

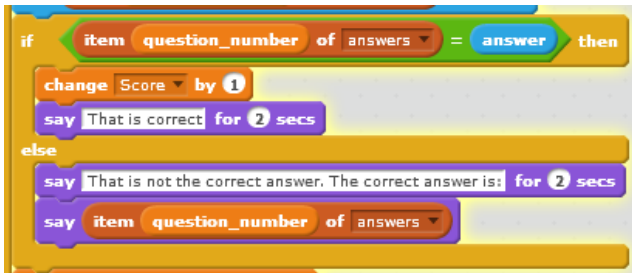
Rather than try and find the bug now we will continue with the rest of the tests as they may help give a clue to where the bug is.

Test 3 appears to work correctly. It displays the correct score and gives the answer when the incorrect answer is given. However when we start the next run through of the game the previous incorrect answer is still displayed.

Test 4 fails. The first two do not display the correct answer and the final answer is still showing when we start the game again.

The correct answers all appear to work OK, but there is a bug with displaying the correct answer when a wrong answer is received from the player.

We can start by looking at the code responsible for displaying the responses to the player.

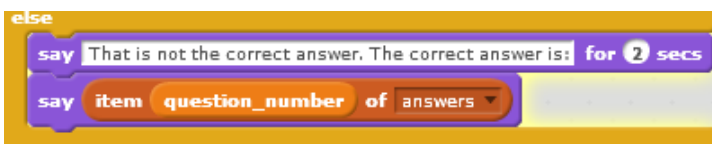


We still need to keep an open mind as often bugs can be a result of code prior to where the symptoms are seen, but it's a good place to start.

Using logical reasoning we know that the answer is shown if it's the last question that is incorrect. As we don't do anything differently for the last question (it uses the same block of code as the previous code) we can deduce that the other questions may still be issuing the message, but that it is disappearing before it is possible to read (or even see) it.

Looking at the part of code responsible for issuing the message and we can see that the first message which we do see "That is not the correct answer. The correct answer is:" has "for 2 secs" in the code whereas the actual answer does not. We can deduce that when that line is run it displays the message, but that it disappears again before it is seen. In the case of the last question it is not hidden because we have then exited out of the loop.

We can therefore replace the say block with:



We can then re-run all the tests which should all work.

Note that we still re-run the tests that worked the first time in case our change unwittingly changed something that was working before.



The tests are not fully comprehensive in that not all possible situations are tested. Even with automated testing it would not be possible to fully test every possible string combination. Testing is normally done on a manageable number of different combinations to try and capture some of the most common occurrences, which is

considered sufficient to capture most bugs.

Other debugging techniques

Fortunately the bug in this program was fairly easy to find, however that is not always the case. One thing that can be an issue is that the bug can be in a different part of code to where the symptom is, or the mistake may not be so easy to find.

If the bug is not easy to find then the following techniques may also help:

- Logical reasoning (work through the program step-by-step in your mind).
- Step through the program (using the single-step mode in Scratch – or add breakpoints).
- Look at the values in the variables (display appropriate variables on the stage as the program runs).
- Write the code out in your own words based on what you see in the program (the opposite of creating the program from the algorithm).
- Create a flow diagram of the expected flow of the program.

Bugs in software are often caused by common mistakes. As you become more experienced with writing and debugging programs then you become more used to recognising common errors and it makes debugging easier.

It is also useful to have a second person help look for bugs. It is easy to think in terms of what you expected the code to do rather than what you have actually created, which is something that may be easier for a second person to see.

Telling the player how they did

Now we have established the program is processing the user input as we expect, we can now provide additional output back to the user telling them how well they have done. In this case we will 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 in the second instance (as we used previously to represent the question number) so that we can include the actual score in the response to the player.



Leave the stage

Finally we need the quiz master to leave the stage.

We would like them to leave from the same side as they arrived so they need to walk in the opposite direction. This could be achieved by creating another costume with the direction changed, but can also be achieved by making the sprite face the opposite direction.

In this instance we will use the same costume, but change the direction of the sprite. This will also mean that the sprite will move forward rather than having to give them a negative direction.

The code block we will use is point in direction:



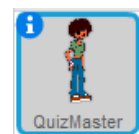
Zero degrees is pointing upwards so set the value to -90 which is facing to the left. Put the block onto the scripts, but don't join it to the existing code yet. Double click to see if it works.



Oh dear!

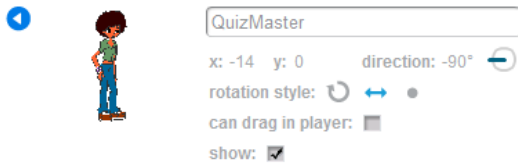
Not quite what we wanted. The character has been turned around to face to the left, but is now upside down.

This is corrected by changing one of the properties for the sprite. Click on the i button on the top left hand corner of the sprite.

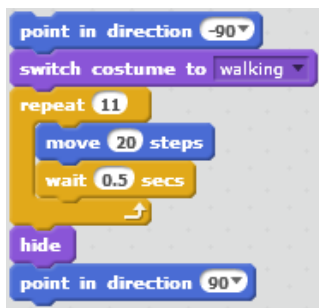


The setting is “rotation style”. There are three options for the rotation style. The left most one allows the sprite to rotate (which is what has happened here), the middle one (arrows pointing left and right) only allows the sprite to flip side-to-side (which is what we want here) and the bottom does not allow any rotation, the sprite will always look to the right.

Click the second button button and that looks much better.



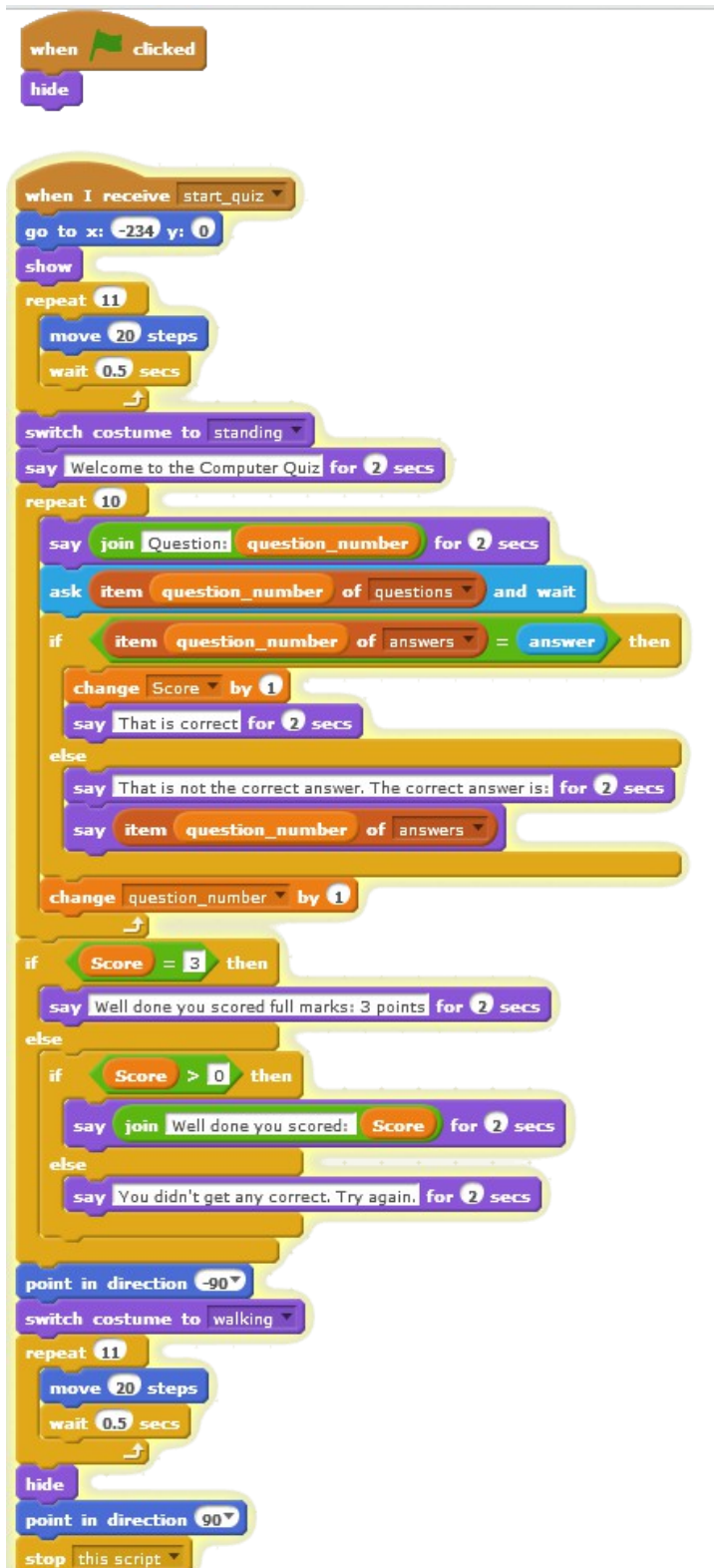
We can now change to the walking costume and use the same movement code we used earlier to make the Quiz Master walk off the stage. Finishing off by hiding the sprite before turning them to point the correct direction for the next game to start.



We can then add this so the existing script and add a stop script to show that we will complete that script.

The completed script for the Quiz Master sprite

After adding the final bit to the script the completed QuizMaster script is as shown below:



You should now be able to test the entire game. Remember to run through all four tests to make sure it behaves as required.

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.

I have already uploaded this game at:

scratch.mit.edu/users/penguintutor/

Whilst the completed code is quite long and at first would have looked overwhelming, by creating it block-by-block you should see that it is fairly easy to understand how it works. We have covered many of the techniques used to make games in Scratch and you should now know enough to create your own games..

Next steps

Whilst we can feel proud about the quiz we've made so far there are still some areas where it could be improved. Here are a few ideas:

- Have appropriate music playing as the quiz master walks on and/off the stage and when the results are announced.
- Make the walk more realistic by having the character's legs move backwards and forwards. The boy3 and girl3 sprites are available with the character in various walking positions.
- 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/teachers
Accompanying website for this guide.
- www.computingschool.org.uk
Organisation supporting computer science education.
- CASPrimaryComputing
Computing at School guide for primary school teachers.
- 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

Many of these resources will be based on Scratch 1.4, especially those based on the Raspberry Pi. The code examples should still work the same, but the Scratch 2 programming environment is different and some of the options will look a little different.

Copyright

This guide is provided through a creative commons license - Attribution-ShareAlike 3.0 Unported.

License details: <http://creativecommons.org/licenses/by-sa/3.0/>

