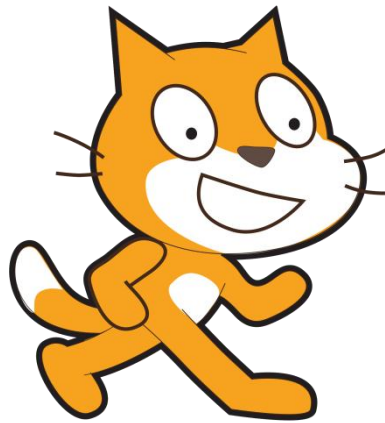


Starting from

SCRATCH



An Introduction to Computing Science

by Jeremy Scott

LEARNER NOTES

Acknowledgements

This resource was partially funded by a grant from Education Scotland. We are also grateful for the help and support provided by the following contributors:

Cathkin High School
Linlithgow Academy
Perth High School
George Heriot's School
Stromness Academy
CompEdNet, Scottish Forum for Computing Science Teachers
Computing At School
Professor Hal Abelson, MIT
Mitchel Resnick, MIT
Scottish Informatics and Computer Science Alliance (SICSA)
Edinburgh Napier University School of Computing
Glasgow University School of Computing Science
Heriot-Watt University School of Mathematical and Computer Sciences
University of Edinburgh School of Informatics
Robert Gordon University School of Computing
University of Dundee School of Computing
University of Stirling Department of Computing Science and Mathematics
University of West of Scotland School of Computing
International Olympic Committee
ScotlandIS
Turespaña
Brightsolid Online Innovation
JP Morgan
Microsoft Research
Oracle
O2
Sword Ciboodle

The contribution of the following individuals who served on the RSE/BCS Project Advisory Group is also gratefully acknowledged:

Professor Sally Brown (chair), Mr David Bethune, Mr Ian Birrell, Professor Alan Bundy, Mr Paddy Burns, Dr Quintin Cutts, Ms Kate Farrell, Mr William Hardie, Mr Simon Humphreys, Professor Greg Michaelson, Dr Bill Mitchell, Ms Polly Purvis, Ms Jane Richardson and Ms Caroline Stuart.

Some of the material within this resource is based on existing work from the ScratchEd site, reproduced and adapted under Creative Commons licence. The author thanks the individuals concerned for permission to use and adapt their materials.

BCS is a registered charity: No 292786

The Royal Society of Edinburgh. Scotland's National Academy. Scottish Charity No. SC000470

Contents

Introduction	5
What is a computer?	5
Types of computer	6
Parts of a computer	9
Hardware	10
Software	11
Programming languages	12
Programming in Scratch	13
1: Scratching the Surface	15
All the world's a stage	15
Putting things in order	17
Did you understand?	18
Lazy or smart?	20
2: Story Time	21
Bugs	22
Event-driven programming	25
3: A Mazing Game	27
The Importance of Design	27
4: Get the Picture?	35
Nesting	36
5: Forest Archery Game	43
Variables	46
Summary	49
Scratch Project	51
Congratulations	60

Introduction

You have probably already used several computers today without realising it.

If you have sent a text, been driven in a car, or checked your watch then you have used a computer. The words you are reading now were typed on a computer.

Computers are all around us. Since they affect so many parts of our lives, it is important to understand how they work.

What is a computer?

A computer is a **machine** that carries out instructions given to it by a human. Without instructions, computers wouldn't be able to do anything.

If this is the case, then what makes them special? Well, computers...

- work faster than humans¹;
- are more accurate than humans;
- can store huge amounts of information that they never “forget”.

It might seem that computers can do almost anything. However, here are some other important things to remember:

- Computers don't have brains; they are not cleverer than humans.
- Computers don't have feelings or “common sense”. This means that there are lots of everyday tasks that humans can perform that computers still cannot.

Activity

Write down three everyday tasks that humans perform but computers cannot (or are not very good at).



1. _____
2. _____
3. _____

¹ At the time of writing, a modern personal computer could perform over 100 billion calculations every second!

Types of computer

Computers come in many shapes and sizes. Computers that most people might recognise include:

Desktop

A **desktop PC** (Personal Computer) is designed to sit on top of – or under – a desk and is used by one person at a time. It is powered by mains electricity and made up of separate devices.



Laptop

Laptop computers combine all the separate devices of a desktop PC into one unit. This can be carried around and powered by mains electricity or battery. Netbooks and ultrabooks are just smaller, lighter types of laptop.



Tablet

This has a large, touch sensitive screen which is used with your finger (or sometimes a special pen). It is battery-powered and very portable. Tablets have an on-screen “virtual” keyboard².



Activity



The personal computers shown above appear in order of oldest to newest types. What does this tell you about the kind of computers people want?

² The word “virtual” is used a lot in Computing. It just means “not real” – it’s something that’s been recreated on a computer. Can you think of any other virtual things you get on a computer?

Other computers that may not be as well-known or recognised by most people include:

Mainframe

This is a large computer which can take up an entire room. Many users can use it at the same time, each with their own keyboard, mouse and monitor.

Mainframes are very expensive and need a team of people to run them. They are owned by large organisations that need to store and process huge amounts of information.

**Server**

A server is a computer that provides services for other computers on a network e.g.

- file server (stores users' files)
- web server (serves out web pages)
- mail server (provides email services)

**Games console**

Games consoles are also computers. Most have a disc drive for loading games and a powerful processor to create realistic graphics.

Many games consoles can also connect to the Internet, letting users buy games online or compete with other gamers around the world.



Embedded

Many devices in your home have an **embedded computer** – a small silicon chip that carries out stored instructions. The modern home has over 100 of these “computers”, built into devices like a toaster, stereo, washing machine, fridge, TV, etc.



A modern car may have another 100 or more embedded computers³.

Activity

Write down three devices in your own home that you think might contain an embedded computer (besides those shown above).

1. _____
2. _____
3. _____

Smartphone

“Smart” mobile phones like Android and Apple iPhone are really pocket computers that can also make phone calls. Many smartphones use large touch screens.

This is a good example of **convergence** where technologies that were previously separate are now combined in one device.

**Activity**

Write down three technologies that are combined in a modern smartphone.

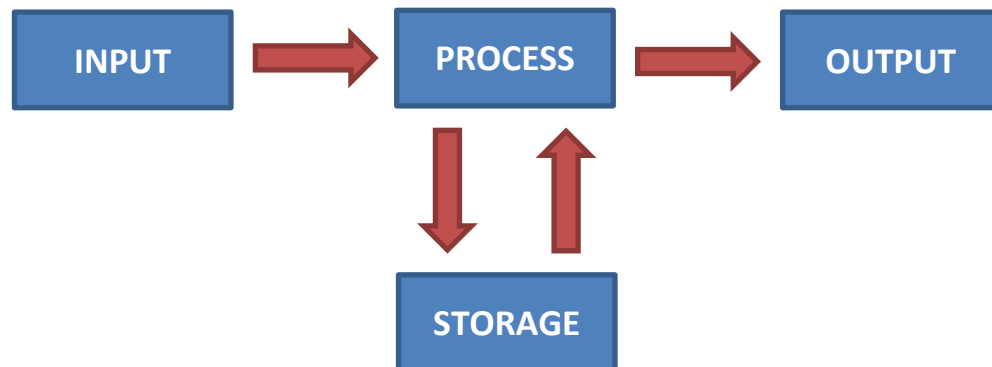
1. _____
2. _____
3. _____

³ Source: <http://www.eetimes.com/discussion/significant-bits/4024611/Motoring-with-microprocessors>

Parts of a computer

A computer is a machine that:

- takes in information
- stores this information
- processes this information
- and gives this processed information back out.



Activity

Write down inputs and outputs for the following activities on different types of computers. When you have finished, create an extra one of your own:



Activity	Input(s)	Output(s)
Playing a video game	Move game controller Click buttons	Character moves Menu selections made
Surfing the WWW		
Making a phone call		
Watching TV		

There are **two** main parts to a computer: **hardware** and **software**.

Hardware

Hardware means **computer equipment**. A single piece of hardware is called a **device**.



There are **four** main types of device in a computer:

Input device	used to put data into the computer
Central Processing Unit (CPU)	where the computer carries out the instructions given by the programs. The faster the processor, the faster your computer will work.
Output device	used for data coming out of the computer
Storage device	used to store programs and data. It is where you save your work on to.

Put simply – if you can touch it, it's hardware.

Activity Decide if the following devices are input, output or storage devices then put each one into the correct column. The first three have been done for you.

keyboard; hard disc drive; monitor; speaker; scanner; printer; mouse; DVD drive; microphone; memory stick; game controller; smartphone touch screen; memory card



Input Device	Storage Device	Output Device
keyboard	hard disc drive	monitor

Software

A computer can perform different tasks, depending on the **instructions** it is given.

A list of instructions is called a **program**. Without a program to tell it what to do, a computer would just be a (useless) collection of hardware devices.

Software is the name given to programs and the information they use.

Activity Complete the table below of ten different jobs you can do on a computer and the name of a software package that lets you do it.

Task	Software package
Browse the World Wide Web	Internet Explorer
Play games	Angry Birds
Edit a movie	iMovie

Programming languages

Computers **follow instructions** given to them by humans. They can solve only the problems that people tell them to solve. To tell a computer what to do, you must know what problem you want to solve and have a plan for solving it.

Unfortunately, these instructions can't just be given to the computer in normal English. A computer can perform tasks very quickly, but it is not intelligent like we are.

A computer will do:

- **only what it is told** and
- **exactly what it is told.**

This means that computer programs have to be written in a very precise way, according to strict rules. There must be no confusion over what instructions mean.

A set of instructions and rules that a program can be written in is called a **programming language**.

Programming in Scratch

The rest of this course will focus on how to write computer programs.



You will be using **Scratch**, created by MIT (Massachusetts Institute of Technology), one of the USA's leading universities.

Scratch is a powerful software development package. It lets you create programs (called **projects**) that combine sound, graphics and animation.

You can upload your projects to the Scratch website and share with other Scratchers around the world. It really is the cat's whiskers!

You will learn how to use Scratch through a series of lessons. At the end of each one, there will be some questions which will help to check if you have understood what you have learned.



A model of the Scratch cat at MIT Media Lab

1: Scratching the Surface

This lesson will cover

- The Scratch environment, including
 - Sprites & stage
 - Properties
 - Scripts
 - Costumes/backgrounds
 - Sounds
- Creating a program with animation & sound



Introduction



Watch the video introduction to Scratch. This will introduce you to Scratch and its screen layout.

http://www.youtube.com/watch?feature=player_detailpage&v=jxDw-t3XWd0



All the world's a stage

A Scratch program contains **sprites** (characters) that “perform” on a **stage**. Sprites and the stage have three kinds of **properties** (or settings):

1. Scripts

These are the **instructions** that control a sprite. Scripts are made from **blocks**.

There are eight different kinds of blocks – to do with motion, control, looks, etc. – and over 100 blocks in total. **Note that sprites need scripts to perform a task.**

2. Costumes/Backgrounds

Costumes are “outfits” for a sprite. The same sprite can have several costumes and so be made to look completely different.

The **stage** can have different **backgrounds** which can be changed. Backgrounds are just like costumes for the stage.

3. Sounds

These are sounds that sprites or the stage can use. Again, each sprite (or the stage) can have many different sounds. Scratch lets you **import** (bring in) recorded sounds or even record your own using a microphone.



Task 1: Up on the Catwalk

Watch screencast **Catwalk**.

This will go over the main elements within Scratch and take you through the task of creating your first computer program. If you get stuck, go back in the screencast or ask your partner.



Task 2: Frère Jacques

Watch screencast **FrereJacques**.

This will show you how to create a simple tune in Scratch. If you get stuck, go back in the screencast or ask your partner.

Did you know...? *Frère Jacques* is one of the best-known songs in the world. It is a French song about a religious monk ("Brother John" in English) who has the job of ringing the morning bell before the days of alarm clocks. Unfortunately, poor Jacques has overslept!



Task 3: My Tunes

Once you have completed Task 2, try creating a program that plays another simple song. Choose one where lines of the music repeat, so you can use the **repeat** command.



Congratulations – you have just started your journey to become a computer programmer!



Putting things in order

Blocks **in the same script** get executed (carried out) **in sequence**, one after the other.

Blocks **in separate scripts** can sometimes be executed **at the same time**. This is called **parallel processing** – having the computer do more than one thing at a time.

For example, if you have several  scripts, they will all get executed **together** when the green flag is clicked.

Extension 1: Dance away

Try to make a sprite dance in time to your music, starting the program when the green flag is clicked. There are **two** ways you could do this:

- create a **single script** that includes the sprite movement blocks amongst the **play note** blocks
- have **separate scripts** for the same sprite – one script plays the tune whilst the other makes the sprite dance.



You can find another screencast (Dancing Queen) to give you some inspiration at <http://info.scratch.mit.edu/node/164>.

Make sure you create a tune, rather than just use a music loop, though!

Extension 2

Experiment by adding some other blocks to your program, such as the looks blocks e.g.



These let you create some really fun effects!



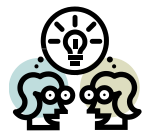
Did you understand?

- 1.1 Look at the section of code opposite that controls a sprite. Write down what you think the user will see when the green flag is clicked.



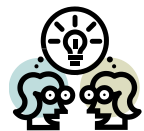
Why?

Now try out the code yourself and see if you were right.



- 1.2 Now add a **wait 1 secs** block between the two move blocks. Describe what happens now.

Explain why this happened



- 1.3 Look at the section of code below that controls a sprite.



Write down what you think the user will see when the green flag is clicked.

Why?

Now try out the code yourself and see if you were right.



1.4 In the stack of blocks below, how many times does the sprite move 10 steps?



1.5 A programmer wants the cat to dance to some music. However, the cat doesn't start dancing until **after** the music has finished!



Why is this?

- 1.6 In the example below, a programmer has chosen a piece of music (sound “Xylo1”) to play during a game. However, when the green flag is clicked, the computer just plays the first note of the music – over and over again!



What mistake has the programmer made?

- 1.7 In **Extension 1: Dance Away**, you made a sprite dance to a tune you created. There were **two** ways you could do this:

- have a **single script** with the movement blocks amongst the play note blocks
- have **separate scripts** for the same sprite – one script plays the tune whilst the other makes the sprite dance.

Why do you think experienced programmers would use **separate scripts**?

- 1.8 Make up a question like those from 1.1–1.6 and pass it to your neighbour.



Lazy or smart?

Computer programmers always look for **shortcuts** to make their life easier.

A good example is how we used a **repeat** block in Frère Jacques to repeat the same line of music instead of having two identical sets of blocks. As well as looking neater, it also means that you won't make a mistake when creating a second set of blocks.

Do you think this makes programmers **lazy** or **smart**? (**Hint**: the answer is smart!)

You can make your life easier, too by spotting shortcuts like this.

2: Story Time

This lesson will cover

- creating stories and plays
- sequencing instructions
- events
- the broadcast command



Task 1: A bad joke



Watch screencast **BadJoke**. This shows how to use Scratch to create a joke or play between two characters.

Once you have done this, try creating a joke of your own – for example, a “Knock, Knock” joke – that uses two characters like the one in the example.

Pay attention to when each character (sprite) “speaks” by planning out the code, including speaking and waiting, like the one below.

Girl	Boy
Say “Hey, I’ve got a joke!” for 3 secs	Wait 3 secs
Wait 3 secs	Say “Okay - let’s hear it!” for 3 secs
Say “My dog’s got no nose” for 3 secs	Wait 3 secs
Wait 3 secs	Switch to costume of boy shrugging Say “How does it smell?” for 3 secs
Say “Terrible” for 2 secs	Wait 2 secs
	Switch to costume of boy laughing Say “<Groan>” for 3 secs



2.1 Write down any problems you had and what you did to overcome them.

Task 2: A short play

Write a short story or play. There should be **two or three scenes** (backgrounds) where the actors (sprites) change costumes.

Keep it simple with only two or three actors (sprites). Write a script on lined paper, with each actor's lines side-by-side, as shown in the previous example.

Hint: You can use the **broadcast** block to let a sprite trigger an event, such as a scene change e.g.

In the sprite script	In the stage script
	



You can find another screencast (Haunted Scratch) to give you some inspiration at <http://info.scratch.mit.edu/node/165>

Extension 1: A walk-on part

Make your characters walk on to the screen and stop at a certain point during the play.

Hint: you will have to start your sprite actors at the edges of the screen and use the **show** and **hide** blocks to make them appear at the correct place every time.



Bugs

A **bug** is an error which stops your code working as expected. There are **two** main types of bug which can occur in a program:

- **Syntax error**
This happens when the rules of the language have been broken e.g. by misspelling a command. Syntax errors usually stop the code from running. Languages like Scratch provide code in ready-written blocks, so you won't make many syntax errors.
- **Logic error**
This means your code runs, but doesn't do what you expect. Unfortunately, it's still possible to make logic errors in Scratch!

Finding and fixing these errors in a program is known as **debugging**.

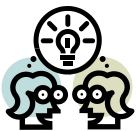
Did you understand?



- 2.1 The program below shows the scripts for two sprites to tell a joke to each other. Why would this program not work?



Girl	Boy
<pre> when clicked say Knock knock! for 3 secs say Doris, for 3 secs say Doris locked. That's why I'm knocking! for 3 secs </pre>	<pre> when clicked say Who's there? for 3 secs say Doris who? for 3 secs say GROAN! for 3 secs </pre>



- 2.2 The program below shows the scripts for two sprites to tell a joke to each other. Aside from being a terrible joke, what is wrong with this program?



```

when clicked
say My dog's got no nose! for 3 secs
wait 3 secs
say Terrible! for 3 secs

when clicked
wait 3 secs
say How does it smell? for 3 secs
wait 3 secs

```

New sprite:

Girl

Boy



2.3 The program below shows the scripts for two sprites to tell a joke to each other. Why would this program not work?



Girl	Boy
<pre> when clicked say Knock, knock! for 2 secs wait 3 secs say Doris, for 2 secs wait 3 secs say Doris locked, that's why I'm knocking! for 3 secs </pre>	<pre> when clicked wait 3 secs say Who's there? for 2 secs wait 3 secs say Doris who? for 2 secs wait 3 secs say GROAN! for 3 secs </pre>

2.4 Now make up a “buggy” question of your own and pass it to your neighbour.

Event-driven programming

Some computer programs just run and continue on their own with no input from the user e.g. your program to play a tune.

However, many programs react to **events** (things that happen), such as:

- the click of a mouse or press of a key;
- the tilt of a game controller;
- a swipe of a smartphone screen;
- a body movement detected by a motion-sensing controller such as a Kinect

In Scratch, event blocks have a curved top (sometimes called a “hat”):



Reacts when the green flag is clicked.
Often used to start a program.



Reacts when a key is pressed. Click the small black triangle to select the key you want to detect. Useful for controlling a sprite, or triggering an action.



Reacts when a sprite is clicked. Useful for controlling characters in a program.

It is also possible to create your own events in Scratch using the **broadcast** command.

2.4 Look at the Scratch environment and write down some other **events** or **conditions** that Scratch programs can react to.

Hint: the **Control** and **Sensing** blocks are a good place to start.

3: A Mazing Game

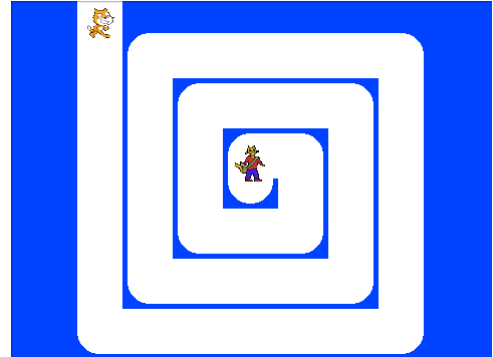
This lesson will cover

- Game creation
- Collision detection

Introduction

You are going to create a simple game where the player guides an “explorer” character around a maze using the arrow keys.

The game will end when the explorer rescues its friend in the middle.



Introduction

Watch screencast **Maze** to learn how to create the Maze game.

Task 1: Setting the scene

Set up the game by importing the stage costume (Maze) and two sprites – an explorer and a friend for the explorer to rescue. **Don't do any more at this point.**



The Importance of Design

Before we make anything – a house, a dress or a computer program – we should start with a **design**. Because there are two important parts to most programs – the **interface** (how it looks) and the **code** – we design these separately.

- The easiest way to design the **interface** is by sketching it out on paper.
- To design the **code**, write out a list of steps it will have to perform **in English**. This is known as an **algorithm** and is just like the steps in a food recipe.

Solving problems like this is what programming is *really* about, rather than entering commands on the computer.

All good programmers design algorithms before starting to code!

Task 2: Designing the solution

Let's look again at the two main things we need to code in our game:

1. moving the explorer
2. reaching centre of the maze (and rescuing the explorer's friend)

The table below shows an **algorithm** for moving the explorer and Scratch **code** that does the same thing.

Algorithm for moving explorer	Code
<p>when the flag is clicked</p> <p>repeat forever</p> <p> if right arrow key is pressed</p> <p> point right</p> <p> move 5 steps</p> <p> if left arrow key is pressed</p> <p> point left</p> <p> move 5 steps</p> <p> if up arrow key is pressed</p> <p> point up</p> <p> move 5 steps</p> <p> if down arrow key is pressed</p> <p> point down</p> <p> move 5 steps</p> <p> if explorer touches the same colour as the maze wall</p> <p> go back to starting position</p>	

Algorithms let programmers concentrate on what the program has to do instead of how to do it on the computer. Once the algorithm is worked out, writing the code is easy!



Notice how an algorithm is **indented** to show which parts belong **inside** other parts e.g.

repeat forever

- if right arrow key is pressed goes inside **repeat forever**
- point right goes inside **if right arrow key is pressed**
- move 5 steps goes inside **if right arrow key is pressed**

Task 2: Designing the solution (continued)

The table below shows an algorithm for the explorer's friend sprite.

From this algorithm, see if you can create the code yourself. **Remember to put it in the friend sprite!**

Algorithm for reaching centre of maze	Code for friend sprite
when the flag is clicked show sprite repeat forever if touching explorer sprite say "Thank you!" hide sprite stop all scripts	Code this one yourself!

Now test your game to see if it works.

Extension 1: Getting in tune

Add a background tune to your game (sound "xylo1" seems to suit, but choose what you think sounds best).



Think about the following:

- Where would be the best place to store this, since it applies to the whole game?
- How will you get the music to keep playing?
- Should you use a **play sound** or **play sound until done** block to play the music?

Extension 2: Add an enemy



Add a sprite that constantly moves back and forth across the stage. If your explorer touches the enemy, the explorer should go back to the start.

Hint: set your enemy sprite to move only left & right.

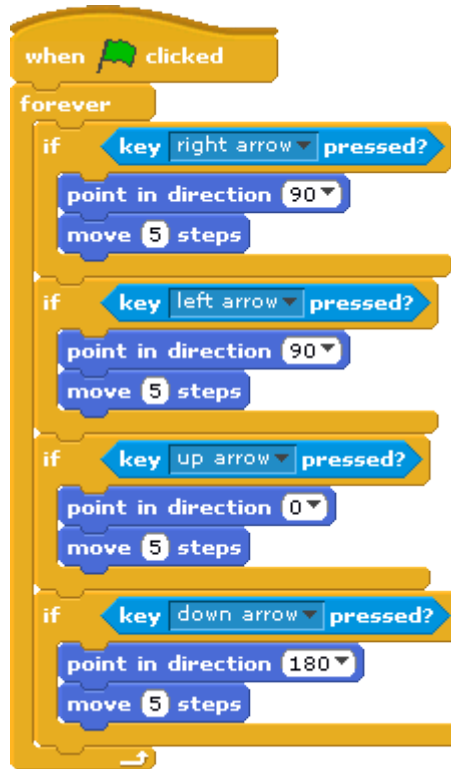
The **if on edge, bounce** block is useful to bounce back and forth off the edge of the stage.





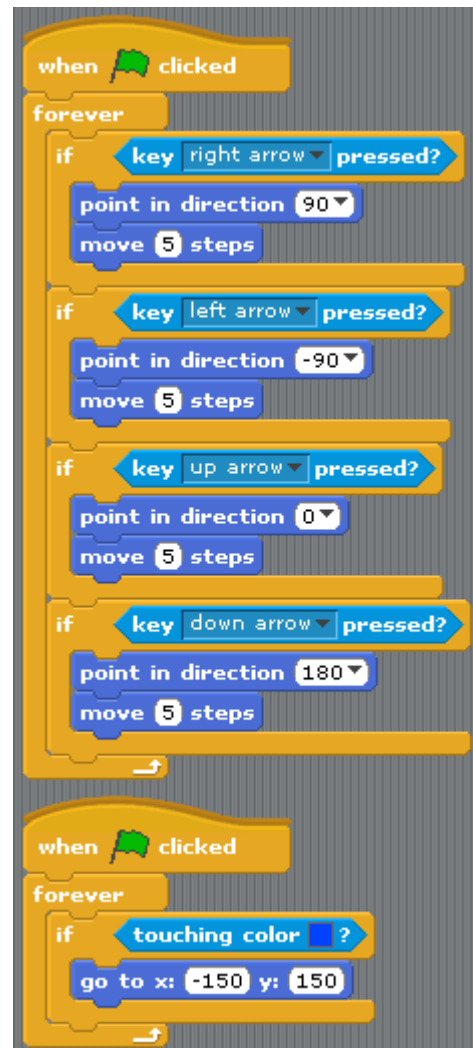
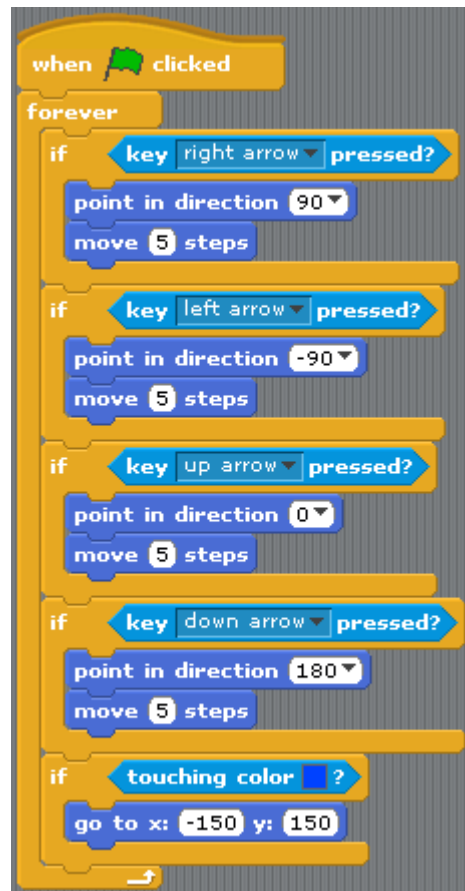
Did you understand?

- 3.1 A programmer creates a maze game like the one you've just created. Unfortunately, her character doesn't move as expected.



What mistake has she made?

3.2 Look at the examples of code below.



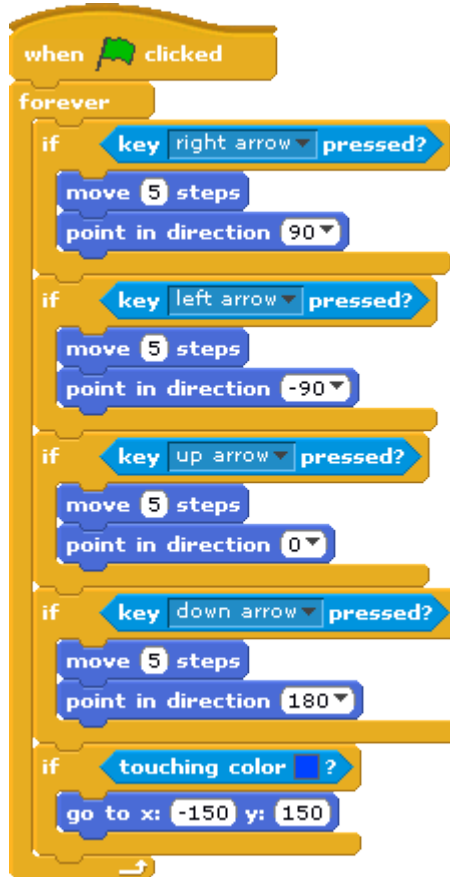
Do they perform the same task? _____

Explain your answer _____



- 3.3 The code below controls a sprite going round a maze. If the sprite touches the side of the maze (the colour blue), it returns to its starting position of -150, 150.

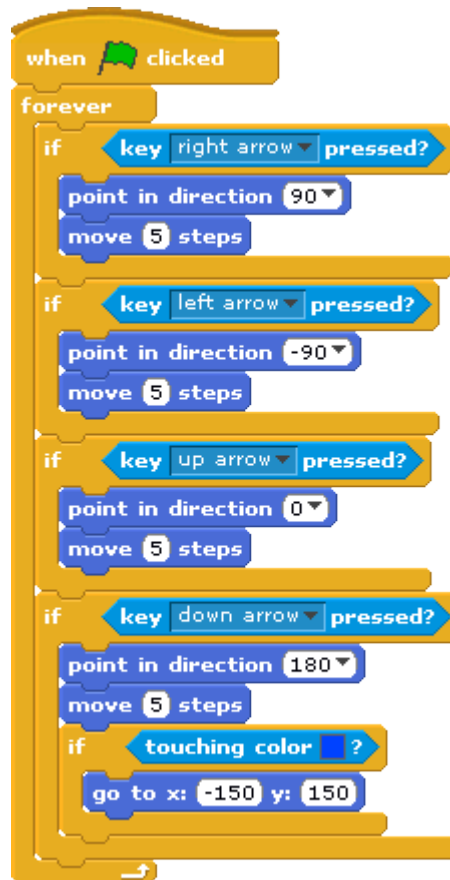
Unfortunately, the sprite sometimes touches the walls of the maze and returns to the start when the player doesn't expect.



What mistake has the programmer made?



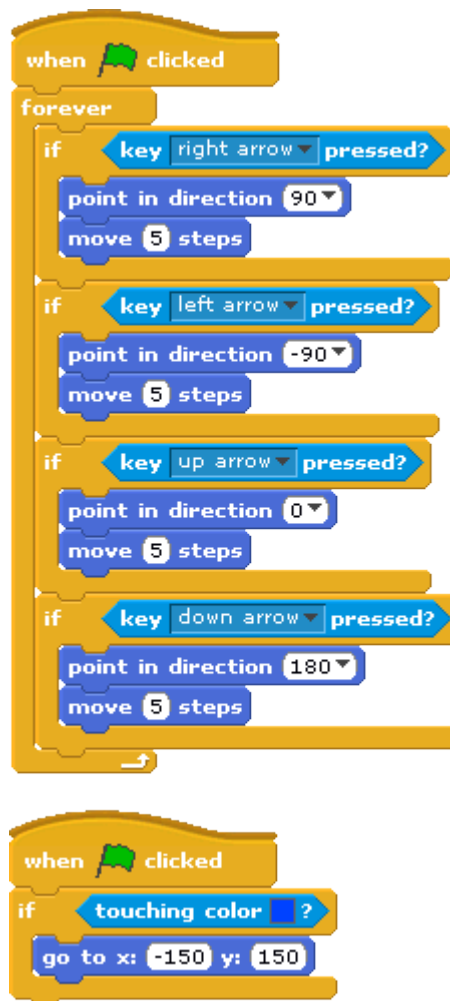
- 3.4 In this example, the sprite is supposed to return to the centre of the maze when it touches the sides (coloured blue); however, it only does this sometimes.



What mistake has the programmer made?



- 3.5 In this example, the sprite **never** returns to starting position, even if it touches the walls of the maze (coloured blue).



What mistake has the programmer made?

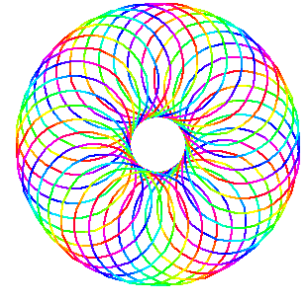
[illegible]

- 3.6 Now make up a buggy question of your own and pass it to your neighbour.

4: Get the Picture?

This lesson will cover

- The Scratch environment
 - Sprites
 - Code blocks
- Fixed loops
- The broadcast and wait command
- Programming computer graphics



Introduction

In this lesson, you will write programs to create simple computer graphics using Scratch's **Pen** blocks.

Task 1: Shaping up



Watch screencast **Graphics**. This demonstrates how to use Scratch to create some simple computer graphics (pictures).

Write down below programs to create the **heptagon** (7 sides) and **triangle**:

Square	Pentagon	Hexagon	Heptagon	Triangle
<pre> repeat (4) move 100 steps turn 90 degrees </pre>	<pre> repeat (5) move 100 steps turn 72 degrees </pre>	<pre> repeat (6) move 100 steps turn 60 degrees </pre>		

Now try out your programs (either double-click on the stacks of blocks or add a **when flag clicked** block at the start).

Did your programs work? _____

If not, why not? _____



The Rule of Turn

Did you spot the pattern here?

In every shape, we turned a **full circle** (360°). To work out how many degrees we need to make at each turn, simply...

Divide the total number of degrees turned in the shape by the number of turns taken

So... in a square, we go round 360° in 4 turns, so $360/4 = 90^\circ$ per turn;

in a pentagon, we go round 360° in 5 turns, so $360/5 = 72^\circ$ per turn

Task 2: You're a star!

Now use the Rule of Turn above to draw a five-pointed star (opposite).

Hint: Pay **careful** attention to what the rule says!



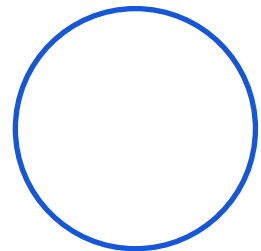
Task 3: Circle

Create a circle. This is easier than you might think: simply

repeat 36 times

move 5 steps

turn 10 degrees



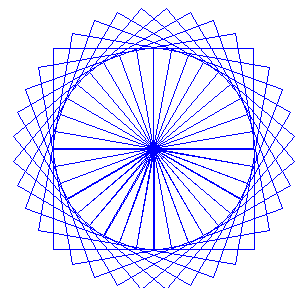
Task 4: Circular pattern

Make a pattern out of 36 squares arranged in a circle of their own.

repeat 36 times

draw a square [put the code to draw a square here](#)

turn 10 degrees



Try changing the shape to squares, triangles or hexagons.



Nesting

In Task 4, we saw one **repeat** loop inside another – this is called a **nested** loop.

In this case, the program starts the outer **repeat**, then enters the inner repeat, which carries on until it's finished. The outer repeat then carries on and so on.

Turn on **single stepping** (Edit menu) to see this happening more slowly. Remember to turn off single stepping when you have finished.

Extension 1: The main event

Create your own **When I receive** scripts to draw each of the shapes you have already created (square, triangle, pentagon, etc.). Remember to use **broadcast and wait** to trigger the **When I receive** blocks.

Once you have done this, adapt your program for **Task 4: Circular pattern** to use a **broadcast and wait** block for the repeating shape.

Extension 2: Our house

Draw a house like the one shown opposite.

Write an algorithm – that is, plan the steps out on paper – before you try to code this!

You will need to use **penup** and **pendown** blocks.

Hint: Think about how you could use the **broadcast and wait** command to reduce the amount of code you create.

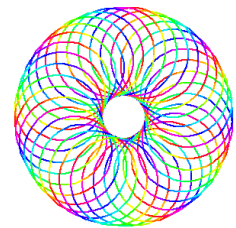


Extension 3: Mmm... doughnuts

Adapt the pattern above to create a multi-coloured doughnut shape.

Write an algorithm before you try to code this!

Hint: There are 36 circles, but the pen moves slightly – with the **pen up** – before putting the **pen down** and drawing the next one. The program also uses the **change pen color by** block to make it colourful.



Extension 4: The Olympic Rings⁴

This is hard! Try to write a program to draw the five Olympic rings. **Write an algorithm before you try to code this!**



Hint: make each circle using a `broadcast and wait` command and think about the spacing between the centre points.

Did you know...? The Olympic flag was flown for the first time at the 1920 Summer Olympics in Antwerp, Belgium and has been flown at every Olympic Games ever since.

The five rings represent the five continents of America, Africa, Asia, Australasia, Europe. The colours – blue, yellow, black, green and red on a white background – were chosen because every nation had at least one of them on its national flag.

⁴ The Olympic rings symbol is reproduced by kind permission of the International Olympic Committee. The Olympic rings are the exclusive property of the International Olympic Committee (IOC). The Olympic rings are protected around the world in the name of the IOC by trademarks or national legislations and cannot be used without the IOC's prior written consent.



Did you understand?

4.1 Look at the program below.

Write down the order in which the scripts are carried out after the green flag is clicked (number them in order 1, 2 and 3).

Number	Script
	<pre> when I receive Square repeat (4) move 100 steps turn 90 degrees </pre>
	<pre> when green flag clicked go to x:0 y:0 point in direction 90 clear set pen size to 2 pen down broadcast Pattern and wait </pre>
	<pre> when I receive Pattern repeat (10) broadcast Square and wait turn 10 degrees </pre>

Now describe what the code will do.



4.2 Look at the code examples below.



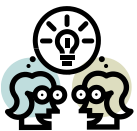
a) How many times will sprite move 10 steps? _____

Why? _____



b) How many times will sprite move 10 steps? _____

Why? _____



4.3 Discuss the following examples from real life. Write an “algorithm” for each one!

a) Getting ready for school



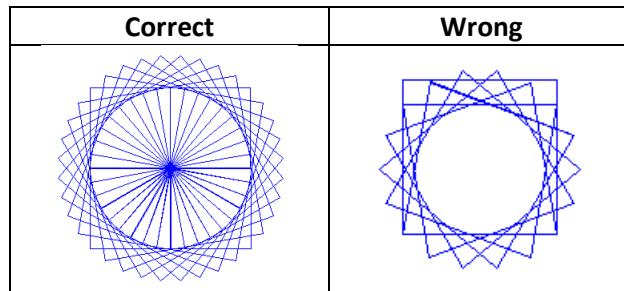
b) Making breakfast



Think: In each example, are there steps that could go in separate scripts and get carried out at the same time?



- 4.4 A programmer tries to create a circular pattern of squares like the pattern labelled “Correct” below. Unfortunately, it always seems to go wrong, displaying the pattern labelled “Wrong”.



Look at the programmer’s code opposite. What mistake have they made?

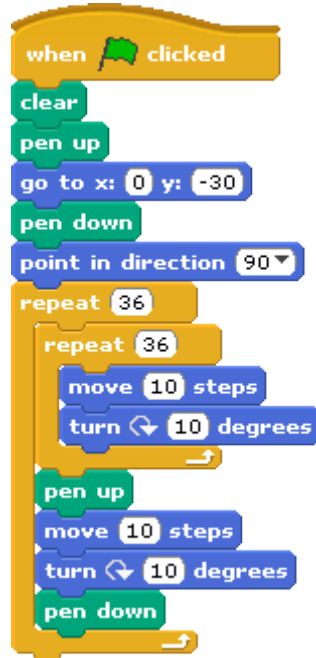
Hint: it’s something to do with how fast the computer works.

- 4.5 Now make up a “buggy” question of your own and pass it to your neighbour.

Did you understand? (Extension 3 only)



- 4.6 A programmer tries to draw a doughnut like the one in Extension 3. Unfortunately, it just draws lots of circles on top of each other.



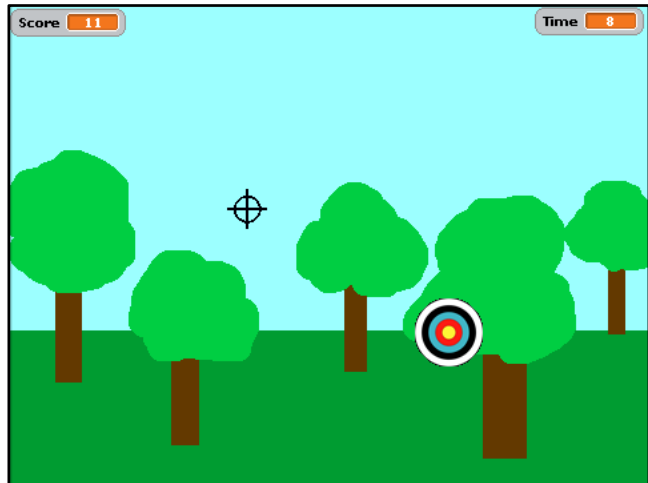
What mistake has she made?

Don't worry if you can't see it straight away – this is tricky! If necessary, enter the script into Scratch and run it to help you understand what's going on.

5: Forest Archery Game

This lesson will cover

- Decision statements
- conditional loops
- variables
- random numbers
- animation
- sound



Introduction

Watch screencast **ForestArchery** to see how to create this game.

Task 1: Designing the solution

Let's look again at the two main things we need to code in our game:

1. moving the target
2. shooting the target

Try to code your program from the algorithms given overleaf, rather than looking at the screencast again.

Algorithm to move target (in Target sprite)

when flag is clicked

repeat forever

glide in 1 second to a random position*

* x is a random number from -240 to 240

y is a random number from -180 to 180

Algorithm to move sight and shoot (in Sight sprite) /...

Algorithm to move sight and shoot (in Sight sprite)

```
when flag is clicked
repeat forever
  go to mouse location (the mouse x and mouse y positions)
  if the mouse button is down (the user has clicked the mouse)
    if the sprite is touching the target sprite
      add 1 to score variable
      play Pop! sound
      Say "Hit!" for 0.5 seconds
```

Task 2: Hit and miss

Change your code to make the program count **misses** as well as **hits (taking off 1 point from the score)**:

```
If touching target
  change score by 1
  play Pop! sound
  Say "Hit!" for 0.5 seconds
else
  change score by -1
  play sound
  say "Miss!" for 0.5 seconds
```

Task 3: Against the clock

Add a timer **variable** to your program which makes the game last 30 seconds. Make the variable appear on the screen as it counts down from 30 to 0.

```
when flag is clicked
repeat 30 times
  wait 1 second
  change time by -1
stop all scripts
```

Task 4: Bullseye!

Using **if** and **touching colour** blocks, change the program so that when the target is hit, it adds the following to the score:

- White – 1 point
- Black – 2 points
- Blue – 3 points
- Red – 4 points
- Gold – 5 points (and says “Bullseye!”)

Task 5: Stay positive!

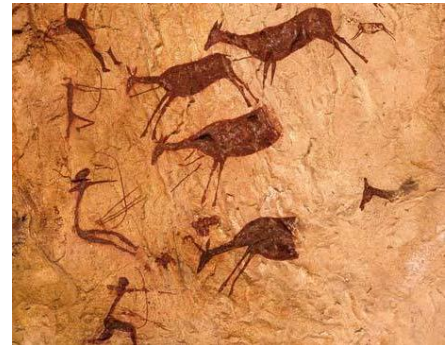
Adapt the program so that the user will **never** get a negative score.

Hint: take off a point only if the score is above zero.

Did you know...? Humans are known to have practised archery for at least 10,000 years. It was first used for hunting (see cave painting opposite⁵), then in warfare.

In medieval England, it was compulsory for all men to practise archery regularly, so they would be skilled if required to go to war.

Nowadays, archery is a popular leisure activity enjoyed by people all around the world.



⁵© Instituto de Turismo de España (TURESPAÑA). Image of cave painting from Cova dels Cavalls remains the exclusive property of Turespaña and cannot be used or reproduced without Turespaña's prior written consent.



Variables

In this game, we introduced the idea of keeping a score using a variable block.

A **variable** is a space in a computer's memory where we can hold information used by our program – just like storing things in a box.

We should always give a variable a sensible **name** that tells us **what kind of information is stored in it** – just like putting a label on the box to tell us what's inside.

To create a variable in Scratch, we **make a variable** block.

Once a variable is created, the information stored inside it can be **set** or **changed** (that is, varied – hence the word “variable”).



Extension 1: A Mazing cool feature

We're now going to add a new feature to your **Maze** game from lesson 3 – a **timer** that gives the user 30 seconds to finish the game.

To do this, add a variable called **time** and create a new script that does the following:

When green flag is clicked

set variable (time) to 30

repeat until time = 0

wait 1 second

subtract 1 from variable (time)

say "You Lose"

stop all scripts

Before you write this script, think about where might be the best place to put it.

Hint: is it something that applies to a single sprite or the whole game?

Extension 2: A Harder Maze

Now create a maze of your own which has more than one route to the middle.

Hint: Just create a new stage background for this.

Extension 3: Do I get a prize?

Create new sprites in your Mazing game to act as bonuses along the way.

These should disappear (hide) when the explorer touches them and add to a score variable. Be sure to place some of them **away** from the quickest route around the maze to make it more challenging!

Extension 4: Now you see it...

Add some code to your Mazing game that shows and hides your bonus sprites after random times e.g. between 1 and 5 seconds (but experiment to see what works best).



Did you understand?

5.1 Look at the script below to make a timer variable count down from 30 to 0.



Will it work? _____

Explain your answer _____

5.2 Now make up a buggy question of your own and pass it to your neighbour.

Summary

Computing Science concepts

You have also learned about some important ideas within Computing Science:

- What a computer is
- Types of computer
- Hardware
- Software
- Program design, including algorithms
- Bugs

Programming structures/commands

In this course, you have used the following programming features:

- Reacting to events
- Decision-making
 - if
 - if...else
- Variables – for example
 - scores
 - timers
- Loops
 - fixed (repeat, forever)
 - conditional (forever if)
- Collision detection
 - if ... touching
 - if ... touching colour

Scratch has many more commands, but you have now learned enough to go on to the next stage.

Scratch features/...

Scratch features

You have also learned about the following features of Scratch:

- Sprites & stage
- Properties
 - Scripts
 - Costumes/backgrounds
 - Sounds
- Animation
- Graphics tools

You now have all the skills you need to create some really amazing Scratch projects!

Scratch Project

Working in a pair or group, you are now going to **create a Scratch project of your own!**

You may have some ideas already, but programs are normally created in a series of stages:

1. Analyse
2. Design
3. Implement
4. Test
5. Document
6. Evaluate
7. Maintain



Or... **A** **D**ance **I**n **T**he **D**ark **E**very **M**idnight!

Analyse



Working in pairs or small groups, **brainstorm three ideas for your project**. Think of how it might link in with other subject areas you're studying.

Think of the areas you've covered so far...

Is it going to be music or graphics-based? A story? A game?

The Scratch gallery at <http://scratch.mit.edu> might give you some ideas.

1. _____

2. _____

3. _____



Now discuss your ideas with your teacher.

Once you have agreed on your project, describe what it will do below.



Design (Screen)

Make a storyboard of your project.

Your sketch should be labelled to show what is happening and what each sprite does.



Design (Code)

Design the steps for your code (algorithm):

- Think about the steps **each sprite or the stage** will have to perform. Write them in English.
- Think about **variables** your project will use.

Sprite/Stage	Algorithm

Sprite/Stage	Algorithm

- Think about **variables** your project will use.

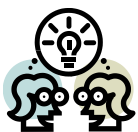
Variable name	What it will store



Implement

Now create your project!

- Gather the **sprites**, **costumes**, **sounds** and **backgrounds**
Remember to give them sensible names.
- Then create the **scripts**
Make sure you have your algorithms in front of you!



Test

Test your project to make sure it works.

Let your classmates test it too and note their comments below:

Good points: _____

Bad points: _____

Describe bugs that were found (by you or by testers) and how you fixed them:

Bug: _____

Solution: _____

Bug: _____

Solution: _____



Document

Let's imagine that you're going to post your project on the Scratch website.

Write down below a brief description (50 words max.) of:

- your project's **main features** and
- **how to use them.**

Remember – you want to get people to try out your project!

Once you have written the description, enter it into your project's notes (**File→Project notes...**).

Evaluate

How did the project turn out **compared to how you originally planned it**?

What **mistakes** did you make on the way?

If you were to start again from the beginning, what would you **do differently**?

Look at your **code** again.

Is there anywhere you could have taken a shortcut to make it “slicker”?

Maintain

What **additional features** would make your project better?



Congratulations

You have now completed this introduction to Computing Science in Scratch!

Remember that you can download and use Scratch at home, so there's no need for this to be the end of your time as a programmer.

<http://scratch.mit.edu>