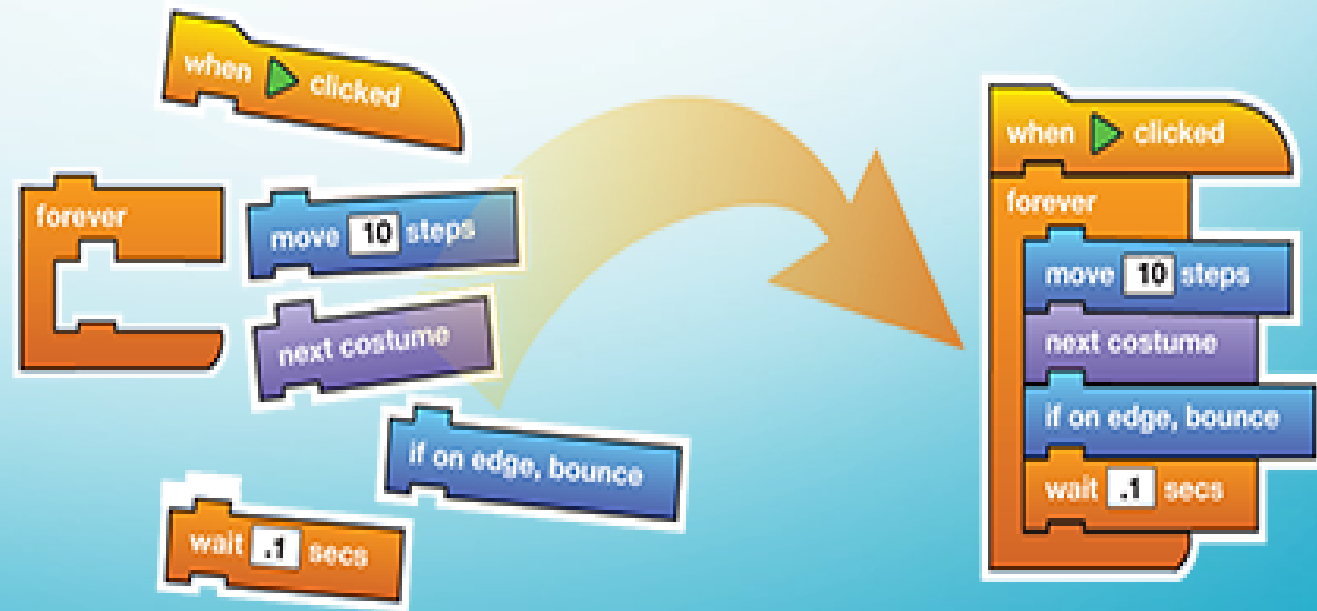



# Creative Computing for All



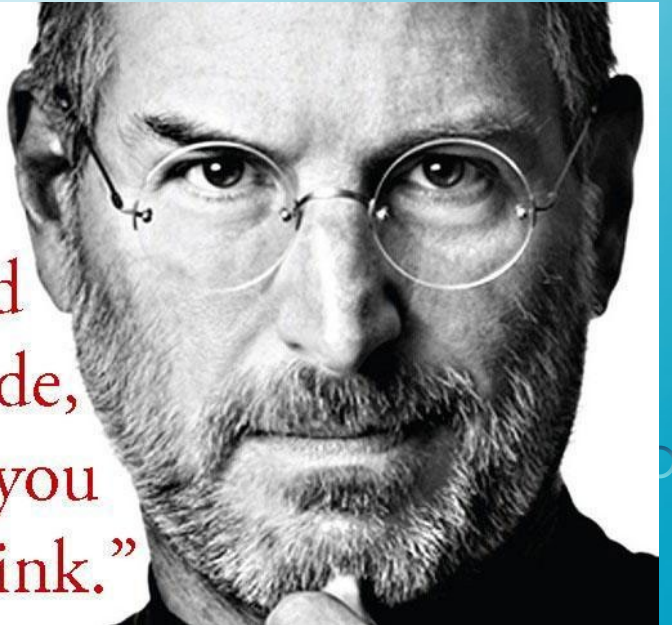
## Computing programmes of study:

*A high-quality computing education equips pupils to use computational thinking and creativity to understand and change the world.*



“In fifteen years we’ll be teaching programming just like reading and writing . . . and wondering why we didn’t do it sooner.”

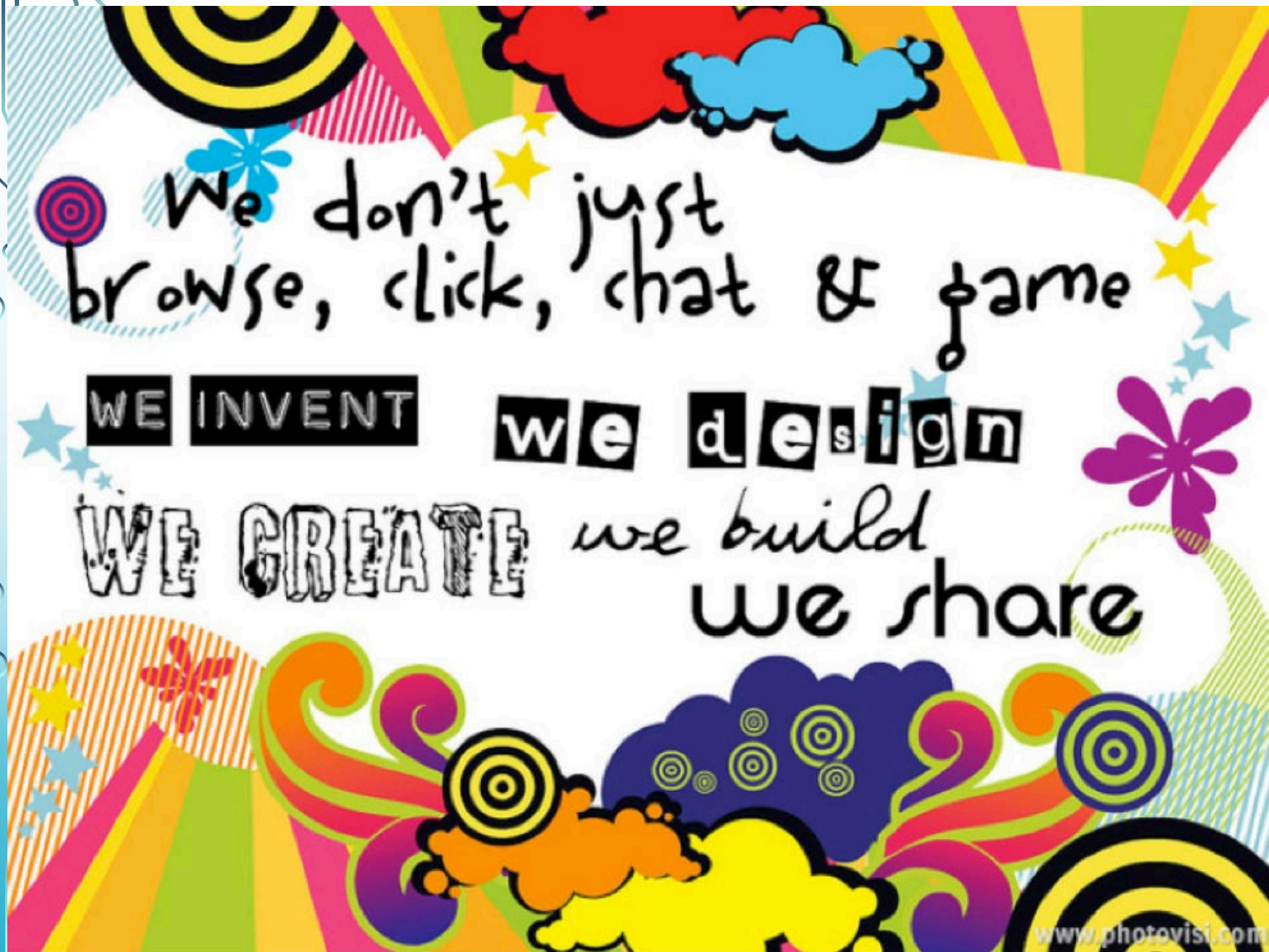
— Mark Zuckerberg



**Steve Jobs**  
1955-2011

“Everyone should learn how to code, it teaches you how to think.”





Move away from consuming ICT to being creators of content: makers, creators, collaborators, digitally critical, responsible and active learners

ICT, Digital Literacy and eSafety are still important parts: not all Computer Science

Three areas: IT, DL, CS

## National Curriculum Framework:

‘ensuring that the national curriculum is taught in ways that enable all pupils to have an equal opportunity to succeed’

- being able to program can be hugely empowering for many with SEN/D.
- some with Asperger’s syndrome find the predictability of computing reassuring;
- the immediate feedback on semantics and syntax in computer code can help some with dyslexia;
- decomposing problems into their components might help some with ADHD;
- simple, text based interfaces can be used effectively by many with sensory or motor difficulties.

<http://milesberry.net/2014/11/making-computing-more-inclusive>

<https://www.gov.uk/government/publications/national-curriculum-in-england-framework-for-key-stages-1-to-4/the-national-curriculum-in-england-framework-for-key-stages-1-to-4#inclusion>

# KEY STAGE 1

- Understand what **algorithms** are; how they are implemented as **programs** on digital devices; and that programs execute by following precise and unambiguous instructions.
- Create and **debug** simple programs
- Use **logical reasoning** to predict the behaviour of simple programs

An algorithm is: -  
a precisely defined procedure  
a sequence of instructions,  
a set of rules,  
for performing a specific task.

Computer programs  
are comprised of a sets of  
rules or instructions



# KEY STAGE 2

Pupils should be taught to:

- design, write and **debug** programs that accomplish specific goals, including controlling or simulating physical systems; solve problems by **decomposing** them into smaller parts
- Use **sequence**, **selection** and **repetition**; work with **variables** and various forms of **input** and **output**
- use **logical reasoning** to explain how some simple algorithms work and to detect and correct errors in algorithms and programs

# PLANNING

Important to remember that the focus of a scheme of work should remain on developing knowledge and understanding of computing concepts *through* activities such as creating games and animations.

It is important that lessons are not driven by the tools available – either hardware or software- but focussed on clear teaching and learning objectives.

# COMPUTATIONAL THINKING

Computational thinking and the concepts behind it, form the basis for much of computer science. Computer scientists are interested in finding the most efficient way to solve problems. They want to find the best solution that solves a problem correctly in the fastest way and using the least amount of resources (time / space).

Is this the most efficient way to solve the problem?

Is this the fastest way?

Does it require the least amount of resources?

Does it solve the problem and give the right answer?

Can it be used to solve other problems?

**Concepts**

**Approaches**



## The Computational Thinker: Concepts & Approaches

### Concepts

**Logic**  
predicting & analysing

**Algorithms**  
making steps & rules

**Decomposition**  
breaking down into parts

**Patterns**  
spotting & using similarities

**Abstraction**  
removing unnecessary  
detail

**Evaluation**  
making judgement



**Tinkering**  
experimenting & playing

**Creating**  
designing & making

**Debugging**  
finding & fixing  
errors

**Persevering**  
keeping going

**Collaborating**  
working together

### Approaches

# Concepts

Logic  
predicting & analysing

**Science:** How does the distance my ball travels change when I throw it harder?

Algorithms  
making steps & rules

**Maths:** How do we calculate an average?

Decomposition  
breaking down into parts

**Science:** Labelling the parts of a flowering plant

Patterns  
spotting & using similarities

**Phonics:** Reading rules: "when two vowels go walking, the first does the talking"

Abstraction  
removing unnecessary detail

**Geography:** Removing detail in the London Underground Tube Map

Evaluation  
making judgement

**PE:** Evaluating a gymnastics routine – what could be improved?

# The foundations of learning at Primary School!

Computational Thinker:  
Concepts & Approaches



**Tinkering**  
experimenting & playing

**Creating**  
designing & making

**Debugging**  
finding & fixing  
errors

**Persevering**  
keeping going

**Collaborating**  
working together

Approaches



## Can computational thinking be meaningful for all learners? Are the P levels relevant to the computing curriculum?

Being able to understand and have some control over the world is hugely important. Technology provides greater opportunities for young people with SEN to access the same things as their peers, and ideally they can begin to have some influence on their experience in school and at home through a better understanding, and ultimately manipulation, of a range of technologies.

P3 ii, 'They apply potential solutions systematically to problems', characterises exactly the sort of pattern recognition and generalisation that lies at the heart of efficient software development,

P4, 'They know that certain actions produce predictable results', captures something of the deterministic, logical nature of computer programs.

We can think in terms of a framework for teachers seeking to develop the beginnings of their pupils' computational thinking.



# Computational thinking as a foundation for programming

Pupils aren't just learning how to use a specific language or piece of hardware, but how to adapt their knowledge for any kind of coding or problem they might meet.

- **Abstractions**

*Ask a pupil 'what did you do at the weekend?' and you don't expect to hear all details, just the essence of what they did. Taking the vital components of a program and disregarding the superfluous is abstraction.*

- **Decomposition**

*If a pupil is going shopping they will need to break down the activity into a number of elements - what do I need to take with me, what am I going to buy, where am I going. Decomposing a problem into smaller parts is at the heart of many activities we do with our students.*

- **Algorithms**

*An algorithm is a sequence of instructions to make something happen, e.g. instructions for making a cake.*

- **Evaluation**

*How can a program/activity be improved and made more efficient? E.g. I was late to school this morning - did I take the fastest route? Maybe I shouldn't buy sweets on the way.*

- **Generalisation**

*Applying rules learnt in one context to another. For example if Mrs X asks me to not to shout out in her lesson, then maybe I shouldn't do that in Mr Y's class."*

# KEY TERMS

- **Decomposition** - splitting a problem into smaller parts
- **Logical reasoning** – predicting what a program will do before you press run
- **Testing & Debugging** - trying it out step at a time, finding and fixing mistakes
- **Abstraction & Generalisation** - which is the information that I need? Finding common patterns and reusing solutions
- **Sequencing** – step by step nature of computer programs
- **Repetition** – loop a number of times or until: ‘repeat’ or ‘forever...until’
- **Selection** – conditional statements such as ‘if...then’, ‘if...else’, **Variables** – used to keep track of the things that can change while a program is running e.g. the score in a game
- **Procedures, algorithms** - program within a program

# KEY TERMS

- **Decomposition** - splitting a problem into smaller parts
- **Logical reasoning** – predicting what a program will do before you press run
- **Testing & Debugging** - trying it out step at a time, finding and fixing mistakes
- **Abstraction & Generalisation** - which is the information that I need? Finding common patterns and reusing solutions
- **Sequencing** – step by step nature of computer programs
- **Repetition** – loop a number of times or until: ‘repeat’ or ‘forever...until’
- **Selection** – conditional statements such as ‘if...then’, ‘if...else’, **Variables** – used to keep track of the things that can change while a program is running e.g. the score in a game
- **Procedures, algorithms** - program within a program

# Logical reasoning

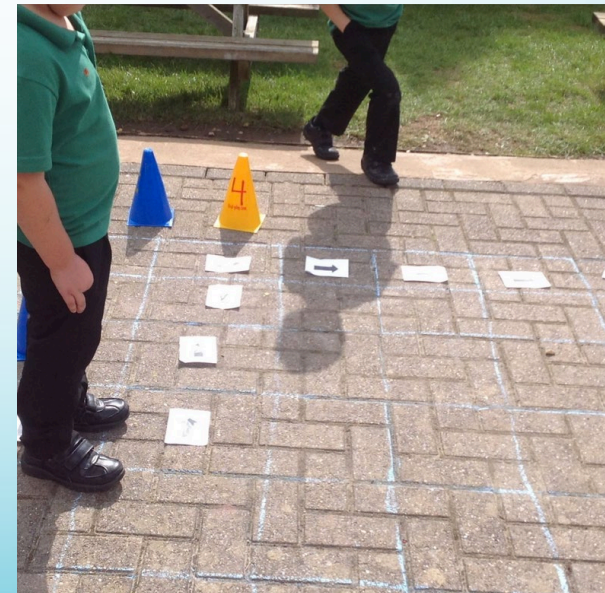
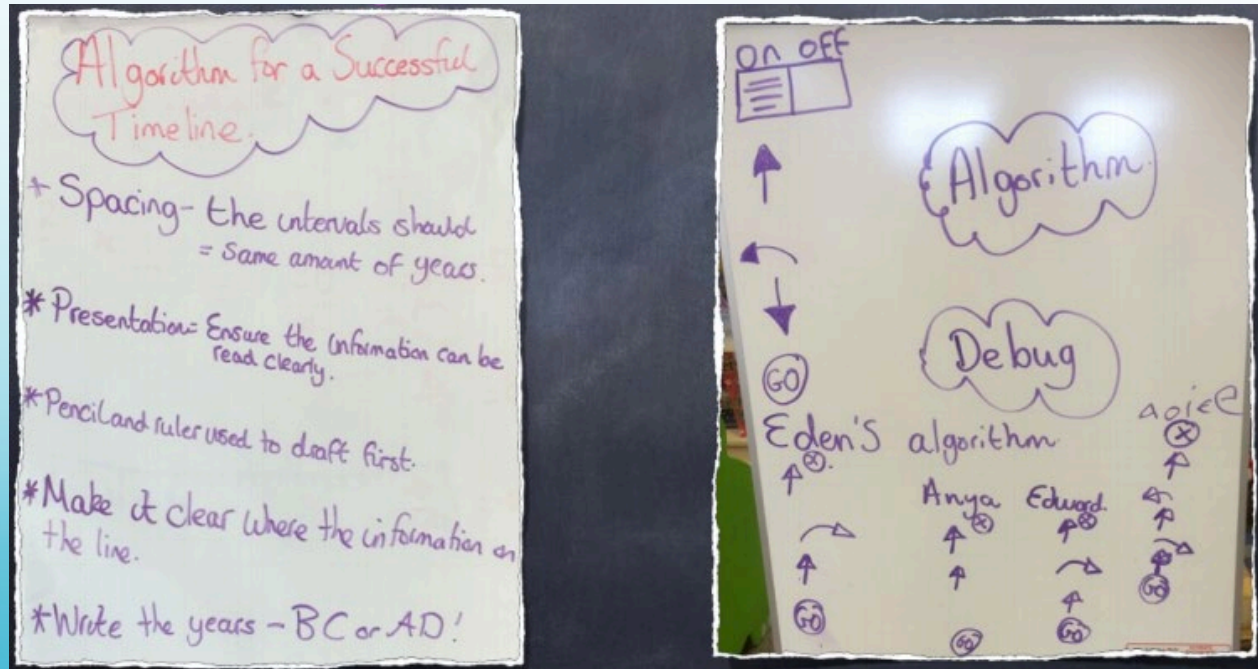
- **Deducing and inferring within Reading**
- **Drawing conclusions from results in Science.**
- **Cause and effect pathways within History.**

*Make this explicit. Then make it transferrable within computing lessons.*



# Algorithms

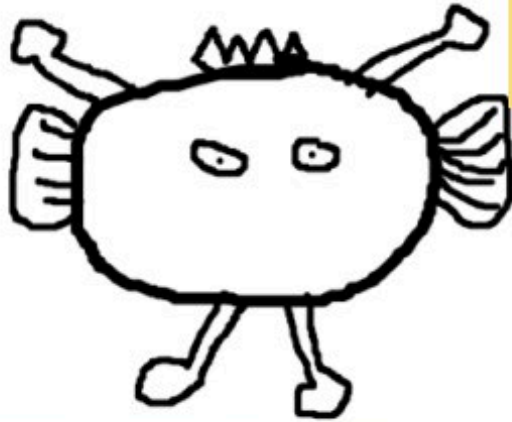
An algorithm is a precise sequence of instructions, or set of rules, broken into steps for performing a task. A program must be 100% exact.





# Algorithms unplugged

Flutty



Can we think of anyway we could change our rules to make sure everyone's looked like this?

- Draw a circle for the body
- Add 2 eyes
- Add a crown
- Add wings
- Add 4 legs.

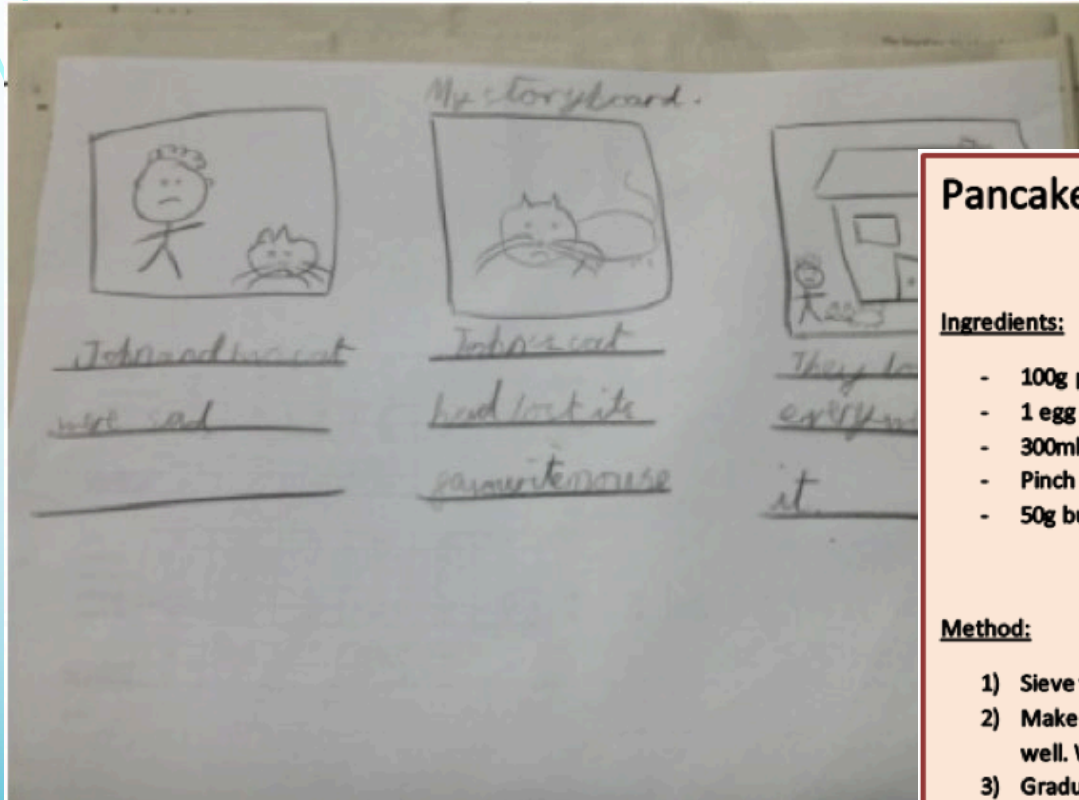
Choose your challenge

**Bobbing-** Add more detail to how big the eyes are and where they should go?

**Flying-** Add more detail to crown and wings. Think about their size and where they are on the body.

**Zooming-** Add detail to all of the instructions. You have two minutes.

# Algorithms across subjects



## Pancake Recipe



### Ingredients:

- 100g plain flour
- 1 egg
- 300ml milk
- Pinch of salt
- 50g butter

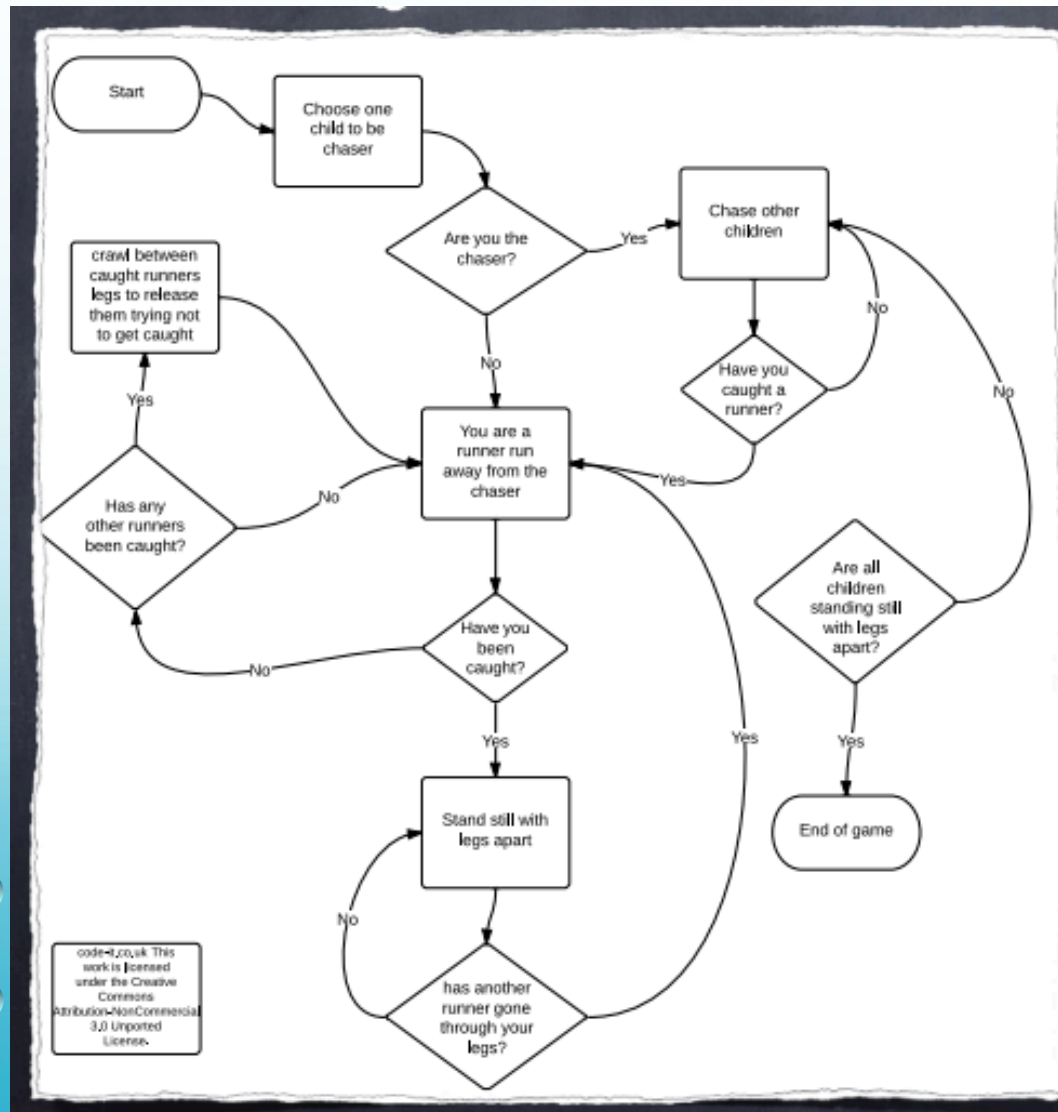
### Method:

- 1) Sieve flour and salt into a mixing bowl
- 2) Make a well in the flour and break the egg into the well. Whisk the egg and flour mixture
- 3) Gradually add the milk and beat to create a smooth batter (consistency of thin cream)
- 4) Heat the butter in a pan. When butter melted, turn heat down to medium
- 5) Coat the base of the pan with pancake mixture (using a ladle is great!)
- 6) Cook for one minutes before flipping the pancake and cooking the other side for 30 seconds
- 7) Enjoy!





# Decomposition and debugging



<http://www.code-it.co.uk/unplugged/playgroundgames/playgroundoverview.html>


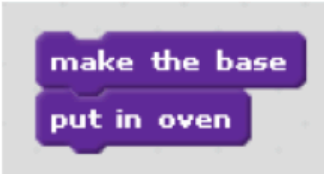
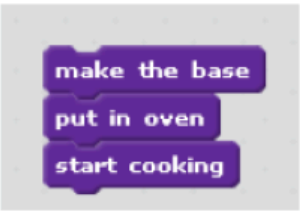
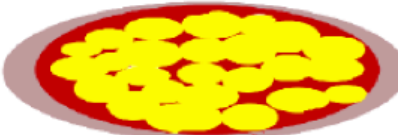
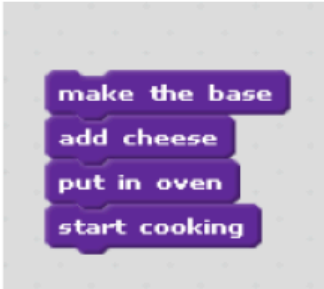
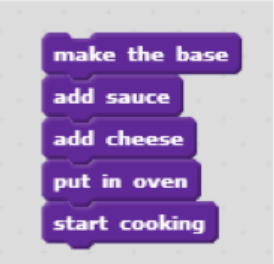
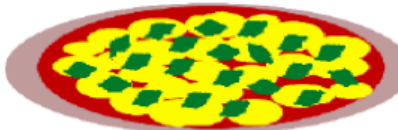
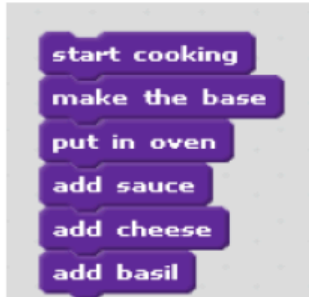
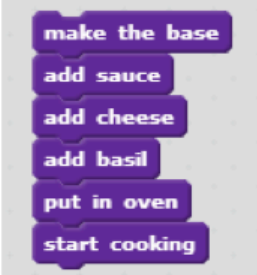
Draw a flow chart to explain how to play Hide and Seek/Tag/Stuck in the mud etc. **Debug** and correct the errors







# Debugging

*KS1: debug simple programs*

*KS1: use logical reasoning to predict the behaviour of simple programs*


Pizza Order <i>What should it do?</i>	Program (with bug)	What does it do? Where does it go wrong?	Fix it! (Your program)
Make 1 plain pizza 		It makes a base and puts it in the oven, but does not cook it	
Make 1 pizza with sauce and cheese 		It makes a base, adds cheese, puts in the oven and starts cooking, but does not add the sauce!	
Make 1 pizza with sauce, cheese and basil 		The steps are in the wrong order!	

# Sequences

Program	Output
	
	

## Story Mountain

Use the Story Mountain to help you plan your stories



Opening

Build up

Problem

Resolution

Ending

[www.commissionforhlt.co.uk](http://www.commissionforhlt.co.uk)

# Selection

Telling the program to do something depending on decisions made: IF... THEN.... ELSE

I am a highly intelligent piece of paper.  
Let's play noughts and crosses.

I am X, and I go first.

**Move 1:** Go in a corner.

**Move 2:**  
IF the other player did not go there  
THEN go in the opposite corner to move 1.  
ELSE go in a free corner.

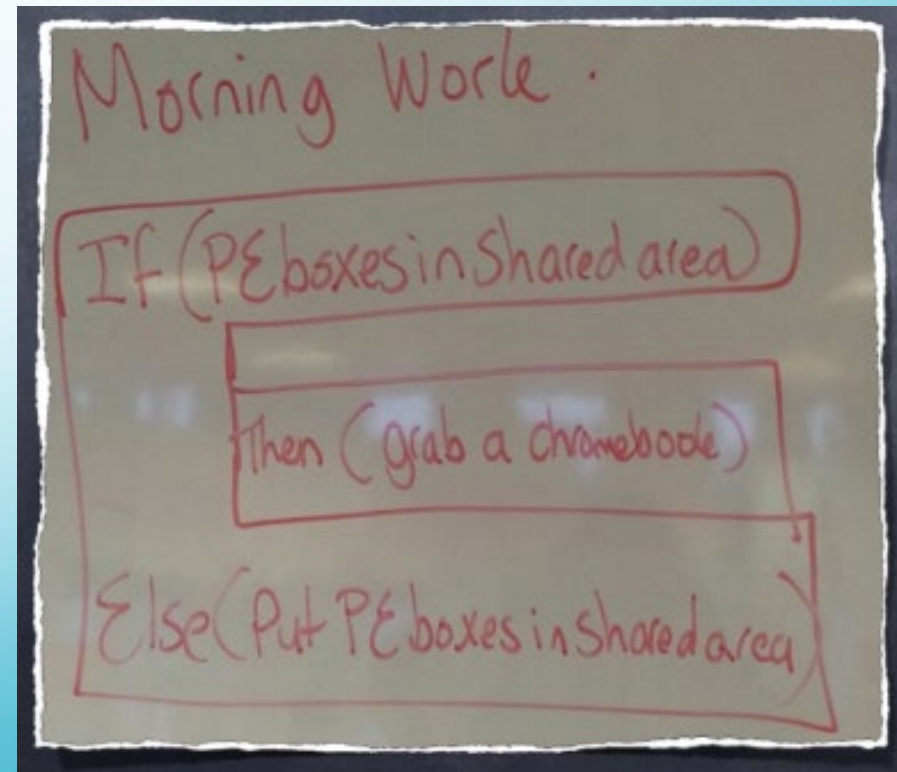
**Move 3:**  
IF there are 2 Xs and a space in a line  
THEN go in that space.  
ELSE IF there are 2 Os and a space in a line  
THEN go in that space.  
ELSE go in a free corner.

**Move 4:**  
IF there are 2 Xs and a space in a line  
THEN go in that space.  
ELSE IF there are 2 Os and a space in a line  
THEN go in that space.  
ELSE go in a free corner.

**Move 5:** Go in the free space.

co <u>i</u> n	bo <u>y</u>
cho <u>i</u> ce	to <u>y</u>
bo <u>i</u> l	en <u>j</u> oy
po <u>i</u> nt	ann <u>o</u> y
so <u>i</u> l	

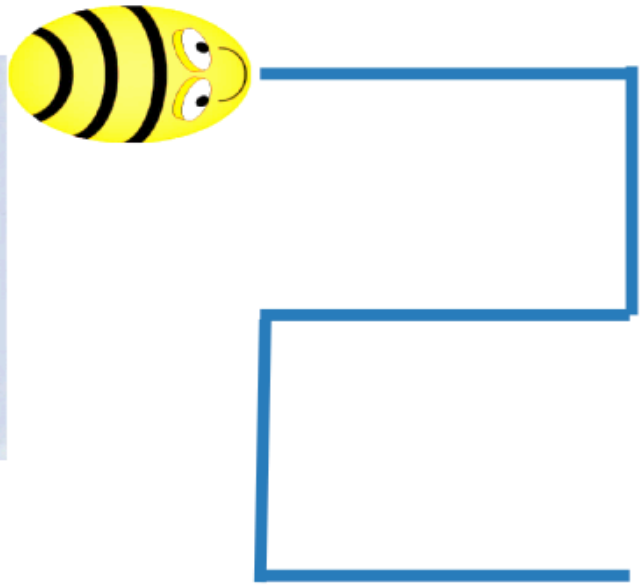
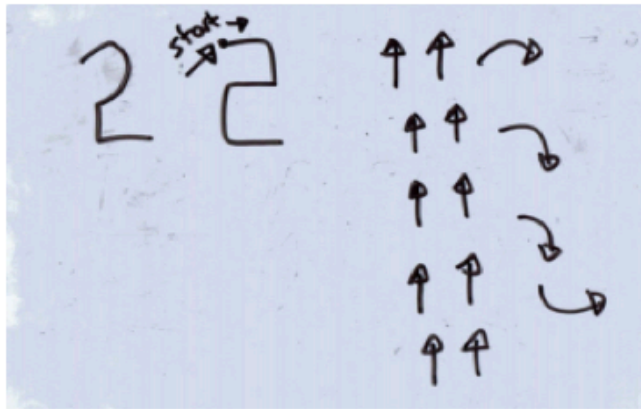
Check through  
grammar...repeat UNTIL  
all correct capital letters are in  
place.





# Understand how algorithms are implemented as programs

**TASK:** Working in pairs can you create algorithms to draw the shape of numerals?

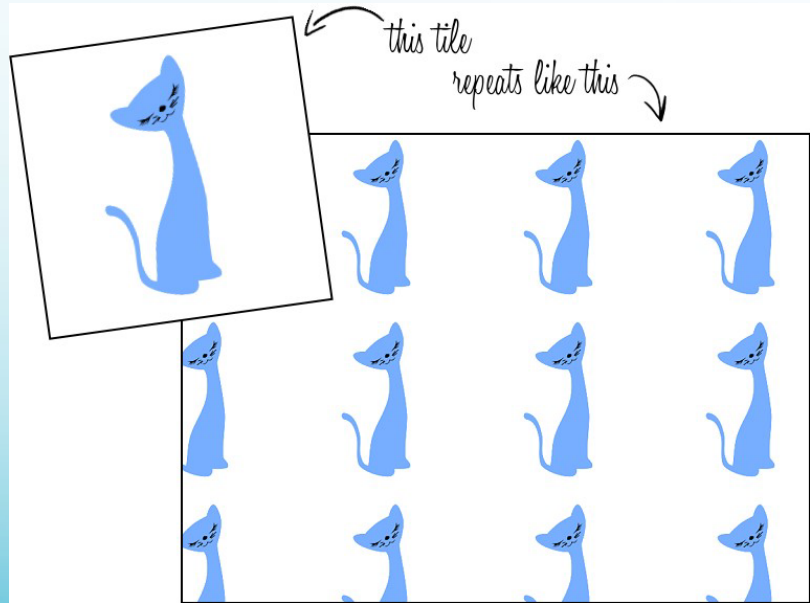


- 1) Write the algorithm using whiteboards/command cards
- 2) Test the algorithm using the Fakebot/other pupils
- 3) Program the Bee-Bot using the algorithm
- 4) Evaluate & debug!



# Repetition

Repeat the same action until the task is done:  
Check through grammar...repeat until all correct capital letters are in place



I know a song that will...  
Get on your nerves  
Get on your nerves  
Get on your nerves  
I know a song that will get on your  
nerves, and it goes like this...



# Variables

Variables store information and 'hold' a value:

The score in a game, lives or health in a computer game, a barrier counting the number of cars in a car park, the cost of items scanned on a till.

Songs with variables:

Ten green bottles

Old Macdonald had a farm



Put a value into the pot which is labelled with its name. Then when it changes the old value is taken out and a new one put in, but the label stays the same.

# UNPLUGGED ACTIVITIES

## Good sources of unplugged ideas:

Code.org <http://code.org/>

Barefoot Computing:

<http://barefootcas.org.uk/activities/>

Computer Science Unplugged:

<http://csunplugged.org/>

Junior Computer Science on Code-it.co.uk <http://code->

[it.co.uk/csplanning.html](http://code-it.co.uk/csplanning.html)

Teach London Computing

<http://teachinglondoncomputing.org/>





# SOME COMMONLY USED RESOURCES

## KS1

- BeeBots
- 2Go
- Logo
- Code.org
- 2Code
- 2DIY
- Kodable
- Alex - app
- Light Bot

## KS2

- Logo
- Blockly
- Code.org
- Scratch Jr
- Scratch
- 2Code
- Flowol
- Tynker
- Hopscotch - app
- Kodu
- Python



# Making programming more accessible

- Pre-loaded Scratch project with comments
- Help sheet
- Paired programming
- Unplugged activities: walking or drawing the shape sequences
- Laminated Scratch blocks
- Using printed Scratch blocks supported by Widgit symbols
- Provide semi-completed programs and debugging challenges
- Use video-walkthroughs
- Limited number of relevant blocks
- Create custom blocks to scaffold a programming challenge
- Notebooks
- Mini plenaries
- Think pair share
- Peer review and feedback
- Provide angles and shapes
- Use other environments: Kodu, Scratch Junior, Blockly, Code.org
- Use Makey Makey, microphone, Webcam or Kinect as input devices