PRIMARY + STEM

Teaching Resource

ACTIVITY: Make me a sandwich

ACTIVITY OVERVIEW

This off-line activity is typically used as an introduction to robotics or coding, but computational thinking can be applied to problem-solving across all disciplines, irrespective of age. It is not a difficult concept to grasp, and as this activity will show, lots of fun to learn!

SYNOPSIS

There is no denying that, in today's innovation-driven economy, skills beyond literacy and numeracy are required for students to thrive. Problem-solving is one of those skills and students who can approach problem-solving positively, creatively, and logically, are more likely to be the problem-solvers, innovators, and creators of this digital age.

Introducing computational thinking to the young is more than laying a strong foundation for programming software. Its relevance in developing thinking and reasoning skills to solve day-to-day problems is what makes this mindset an essential part of every child's education. When writing a narrative, computational thinking takes the form of a writing plan or graphic organiser, where the story is first broken down into different elements. At home, it could come across as deciding what to cook, based on what is in the pantry. When programming computers, computational thinking results in coding.

This off-line activity is typically used as an introduction to robotics or coding, but computational thinking can be applied to problem-solving across all disciplines, irrespective of age. It is not a difficult concept to grasp, and as this activity will show, lots of fun to learn!

Foundation – Year 2

- Follow, describe and represent a sequence of steps and decisions (algorithms) needed to solve simple problems (VCDTCD017)

Year 3 – 4

- Define simple problems, and describe and follow a sequence of steps and decisions involving branching and user input (algorithms) needed to solve them (VCDTCD023)

Year 5 - 6

- Define problems in terms of data and functional requirements, drawing on previously solved problems to identify similarities (VCDTCD030)
- Design a user interface for a digital system, generating and considering alternative design ideas (VCDTCD031)
- Design, modify and follow simple algorithms represented diagrammatically and in English, involving sequences of steps, branching, and iteration (VCDTCD032)
- Develop digital solutions as simple visual programs (VCDTCD033)

ACTIVITY, MATERIALS AND INSTRUCTIONS

Activity

Students are introduced to elements of computational thinking through this fun activity, which can be done in groups with older students, or as a class activity for younger children.

Before embarking on this activity, survey the class for any allergies to the ingredients and make the necessary changes, (e.g. swap Vegemite for hundreds-and-thousands, butter for margarine, etc.).

Materials for 30 students

- A loaf of bread
- 6 servings of Vegemite
- 6 servings of a soft spread, (e.g. butter or butter alternatives)
- 6 x plastic knives
- 6 x plates
- Paper towels
- Enough plates or containers to dish out Vegemite and the spread
- 6 printouts of 'instructions' that has been cut out
- Sufficient pen and paper
- Scissors
- Optional: blindfold, gloves, glue

Instructions

1. Students work in groups of four or five to prepare a Vegemite sandwich.

   Assign roles to students - one student is the sandwich-producing robot, while the others are programmers who will develop and give instructions for the process.

   Variation: For younger students, this could become a class activity where the teacher is the robot. Students still work in groups to create the instructions and the robot-teacher executes the steps from each group.

2. a) With the exception of the robot-students, the rest of the programmer-students come up with a sequence of steps for the robot to follow.

   From the pieces of cut out instructions, students select and sequence an order that makes sense to them. Students should identify the more specific instruction, ignore the irrelevant ones, and find the shortest solution.
   Variation: There is a choice of instructions for younger and older students

   b) Meanwhile, the robot-students gather the necessary materials for the activity and place them on their sandwich-making station. They also wash their hands with soap.

3. Make the sandwich.

   Give students about 10 minutes for this task.

   Rules/Conditions:

   a) Seat the programmer-students a distance away from the robot-student to give the instructions. Programmers cannot be in physical contact with the robot.

   b) We assume that the robot already possesses the necessary sensors, output devices, and programming to see the material, pick up objects, scoop out spreads, and apply the spread.

   c) The robot cannot speak and can only shake its head or shrug its shoulders when it doesn't understand the instructions. The robot must only follow the instructions that are given to it; and can decide whether the instructions are sufficient to carry out the task.

For added challenge, blindfold the robot.

4. Assess the process and make modifications.

   What are some problems that arose? What changes need to be made to the instructions?

5. Repeat the sandwich-making process with the improved instructions.

   There should be marked improvement to the process because of changes made to the instructions. But it also helps that the sandwich-making robot possesses sophisticated artificial intelligence to self-learn and improve each time!

6. The robot eats the sandwich!

7. Discuss the importance of being systematic in this activity, and introduce how the concept of computational thinking can be applied to this i.e.
   - breaking down the problem into smaller parts (decomposition)
   - finding similarities between steps (pattern recognitions)
   - focusing on the important aspects, ignoring the irrelevant details (abstraction)
   - developing a step-by-step solution (Algorithm design)

Which component of this process did students find most difficulty?

HOW TO USE THIS ACTIVITY WITH YOUR STUDENTS

This activity introduces students to the idea of approaching a problem systematically, and there are many simple ways to go about extending student understanding of thinking and reasoning skills related to problem solving from here.

Foundation - Year 2

Introduce students to the idea of computational thinking in our everyday tasks. What are some areas of our everyday lives where it is useful to be systematic, (e.g. getting ready for school in the morning? Preparing a meal?)

With robots and machines, these steps need to be spelt out explicitly so that each task is done properly.

In the area of problem-solving, students are expected to learn how to sequence events and actions, and to do this through providing instructions to programmable devices such as robots. For this age group, use a classroom set of robots that students are already familiar with, or use the simplest programmable robots, such as Beebots, so that the focus remains on the sequencing of events.

The activity could be as straightforward as making the robot navigate a path out of the classroom. Other ideas include using the robots to play naughts-and-crosses, or even using the robots in storytelling, (e.g. where the robot is the Big Bad Wolf that visits the homes of the three little pigs and spins in front of their door). Some school robot kits also come with patterned floor plans that teachers can use to support mathematics learning, by using floor plans with geometric shapes or numbers, or for spelling lessons, by using floor plans with letters to spell a word.

Years 3/4

Use this activity to emphasise how there are many ways of solving the same problem. As groups present their sandwich-making robot, one at a time, other students can take notes of how their own instructions differ. For programming robots and machines, however, it is the simplest working program that is the best solution.

As an extension to this activity, ask students to write out instructions to tie a shoelace or to eat a TimTam biscuit with milk. Students present their solutions and vote on which set/s of instructions are their favourite, or most plausible alternative.

Years 5/6

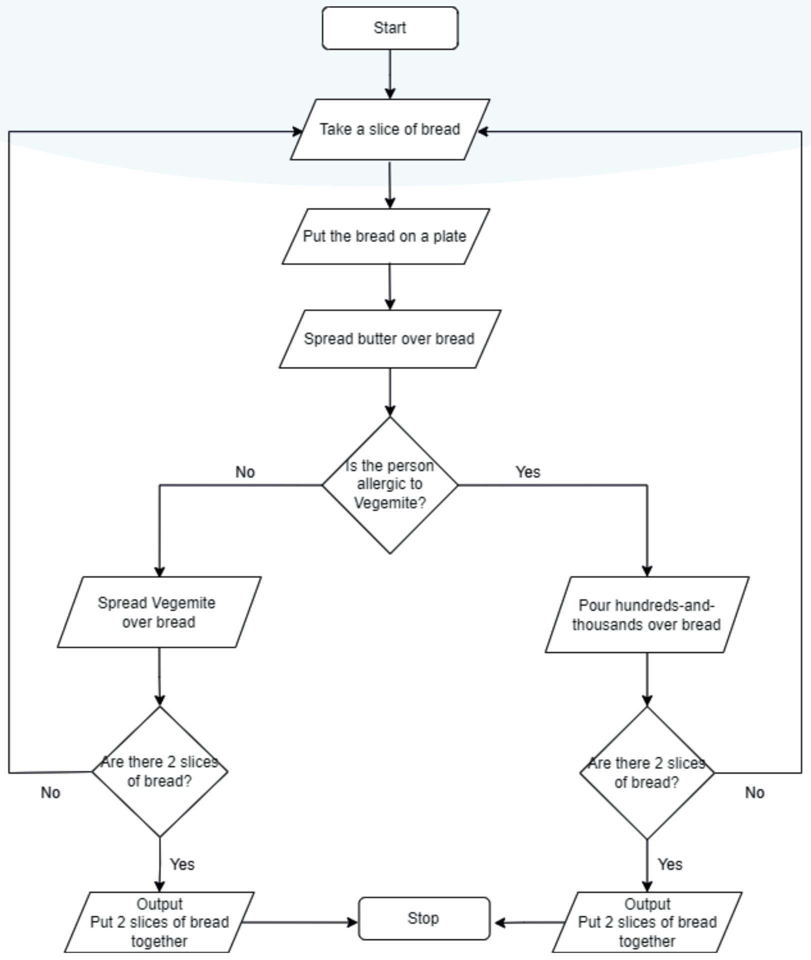
Students should be able to complete this activity quite quickly and easily, so this could serve as a fun introduction to representing information, (e.g. a series of instructions, repetitions within an instruction, or making a decision between two instructions).

Students could represent their set of instructions (algorithms) for making the Vegemite sandwich as flowcharts or tree diagrams, or use words to express their ideas. For variations, apply some conditions for students to include:

- Using loops in a flowchart, or a 'REPEAT' statement if using words, or when they identify repetitive instructions
- There is an unfortunate allergy to Vegemite

Students present their flowchart to their classmates for feedback.

Below is an example of a flowchart made with Draw.io and then simply copied and pasted to a Word document:


Start
Take a slice of bread
Put the bread on a plate
Spread butter over bread
Is the person allergic to Vegemite?
No
Yes
Spread Vegemite over bread
Pour hundreds-and-thousands over bread
Are there 2 slices of bread?
No
Yes
Are there 2 slices of bread?
No
Yes
Output Put 2 slices of bread together
Stop
Output Put 2 slices of bread together

For more information on how to use flowcharts:

- [Using flow charts to design algorithms (education.vic.gov.au)]
- [Representing an algorithm: Flowcharts - Algorithms - KS3 Computer Science Revision - BBC Bitesize]

DISCUSSION SECTION AND KEY THEMES

KEY THEMES

Computational Thinking

The concepts of computational thinking first appeared – thanks to computing pioneers - in the 1950s but only gained a foothold in the computer science educational community in 2006. That was when a professor of computer science proposed that computational thinking was a fundamental skill for everyone - not just computer scientists. She also argued for the importance of integrating these problem-solving skills across other disciplines in school.

Computational thinking describes a structured approach towards problem-solving and forms the framework of how programmers find solutions. However, this mindset is also a powerful thinking and reasoning tool for students to use in every aspect of their lives.

Computational thinking comprises four components:

1. Decomposition

   This describes breaking down the problem into smaller, more manageable parts to achieve the desired solution. Complex problems can often be overwhelming, so breaking them into separate components will make them easier to be understood and solved.

   In English, decomposition is the process when students plan a narrative or persuasive piece, in parts, before they write the actual story.

   Whether using a more traditional plan or a graphic organiser like a story arc, students consider different elements, such as the introduction, the conflict, the climax, and the conclusion, separately, before assembling these ideas to form the story.

2. Pattern recognition

   Pattern recognition is the act of looking for similarities or patterns in a problem, so that predictions and rules can be made to resolve other similar problems more effectively. The predictions and rules allow large quantities of data to be grouped (classified and clustered), based on how they are similar to the types of data that is already in the system - and then allows decisions to be made, accordingly.

   Pattern recognition forms the basis of facial recognition, iris recognition, computer vision, speech recognition, fingerprint identification, and seismic analysis. Teachers and students use pattern recognition skills when sorting living things from non-living things, classifying living things into kingdoms, identifying beats and scales in music, applying mathematical rules to similar situations, and even when identifying words that rhyme.

3. Abstraction

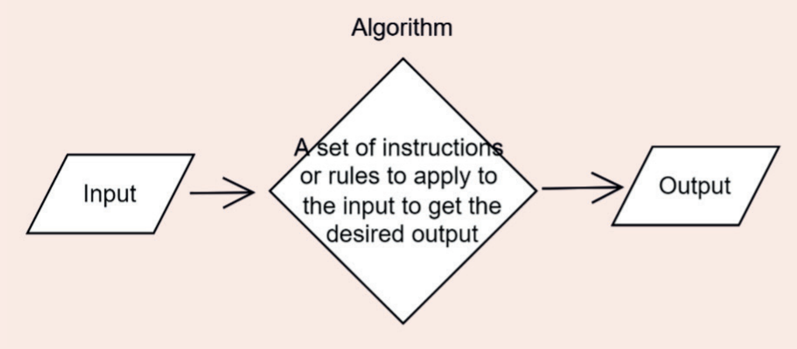
   Abstraction is the process of focusing only on the important parts of the problem - and ignoring irrelevant details - to solve it efficiently. Once we have recognised patterns in problems, abstraction is used to gather general characteristics, and then filter out the details that we do not need.

   Abstraction allows us to have a clearer idea or 'model' of the problem. Having abstracted, we know that all birds have wings, feathers and beaks.

   The colour of the feathers and whether those feathers help the bird fly are details that we can leave out. Abstraction also helps us not to be overwhelmed by details. For instance, when learning to ride a bicycle, we do it by only learning the necessary skills (e.g. balancing, steering, braking, pedalling), without having to know the history of cycling or the physics of motion. At school, teachers and students use abstraction when summarising a story and identifying the main ideas of a lesson.

4. Algorithms

An algorithm is set of step-by-step instructions, or rules, to solve a problem. Each of these steps is simple and well-defined, and ordered to work in a specific sequence. Algorithms are a starting point for creating a computer program and are often presented in a flowchart. An algorithm computes the input according to the instructions and produces the desired output. Like a recipe, an algorithm takes the input (ingredients), applies the set of well-defined instructions, and produces the output (the dish).


Algorithm
Input
A set of instructions or rules to apply to the input to get the desired output
Output

After an algorithm has been established, programmers express these instructions by coding so that the computer knows what to do.

At school, algorithms can be used to improve understanding of grammatical rules when students draw flowcharts and tables to show when specific rules apply. In science, students gain a better understanding how to use the periodic table once they understand the rules of how elements are sorted in the table. In art, students can even follow an algorithm to draw a portrait.

QUESTIONS AND ANSWERS

Can machines program themselves to do things?

Yes – and if machines have the capability to do this, we say that they possess artificial intelligence (AI). Programmers have developed software which allow computers to collect data each time they carry out a task, then analyse the data to perform the same task better the following time. The result is machines that are capable of reasoning, learning, problem-solving, and making quick decisions.

Prominent examples of AI include robots in manufacturing which 'learn' to become more efficient over time, robots in surgeries, and even chatbots to help shoppers get a better customer service experience while shopping online!

Is it difficult to code a program?

As with learning any skill, coding takes effort, time, and expertise to master. There are several languages students can learn to code – some easier than others, such as Scratch, Python, Java, etc. There are many free online platforms where students can find fun ways to learn coding (e.g. code.org, Scratch, Pencil Code, etc.).

Creating a great program takes much more than learning a programming language to tell a computer what to do. The greater challenge is fully understanding what the problem is that one is trying to solve, and the ways to go about solving it (e.g. knowing how to tell a computer what to do – coding - only comes after someone knows exactly the game they want to create).

Can we program a machine to be a human being?

Robots designed to look like humans are called humanoids. This has been a fascination for many people, and robots designed to look like humans have come a long way in terms of how realistic they are becoming. It involves a lot of complex programming to recreate human responses, and expressions. Famous humanoids include Sophia and Ameca. Meet Ameca, the humanoid robot, and the engineering team that made her possible on this show produced by Wired: https://youtu.be/6iO6XhbVQfs

Most experts interviewed by Discover Magazine (2017) on this topic agree that robots are not humans, but also say that with more humanoids being manufactured, we might have eventually to make special considerations for them.

- Do Robots Deserve Human Rights? | Discover Magazine

What do students think?

How can computational thinking help me?

Computational thinking is a logical way of reasoning and making decisions that can help us solve problems every day. It is certainly not the only approach towards solving problems, but it's a logical one. All of us, including adults, benefit from using this way of thinking when we face problems.

Computational thinking can help us when we are overwhelmed with tasks like getting ready for school in the morning on our own. It helps to first break down the big task into chunks – such as wash-up, get dressed, have breakfast, make lunch, walk to school - and deal with these smaller tasks one at a time. It also helps us to disregard any part of the process that is not crucial to getting us to school (e.g. watering plants before heading out, or counting the number of grapes to add to our lunchbox).

Do people need to code everything from scratch every time they want to program something?

Absolutely not. There are free computer programs available on the internet, that anyone can download, modify, and use. These are known as open-source software and are contributed by programmers all over the world, in the spirit of open collaboration (sharing).

For example, if we wanted to make a sonar device scan a room and send this information to our laptop, we would need a program to instruct our computer on what to do with the sonar. Instead of coding instructions to do this from scratch, we can download a ready-made one from a website such as SourceForge and, with some modifications, use it to make our sonar and laptop work as intended.

Do human beings have codes?

The meaning of code in this context is different from coding instructions into computers. But yes, humans are born with a program containing codes from our parents. And no two humans, aside from identical twins, have identical sets of codes.

Humans, and all other living things, have a unique genetic code in the form of DNA.

The genetic code of a living thing refers to the instructions contained in cells which tell it how to make all the parts for creating that unique organism. There are four different types of molecules that make up DNA (like having only four different) and how these molecules are arranged (similar to words) determines the types of proteins that can be made.

In 2003, the Human Genome Project successfully identified and mapped out the complete set of human DNA! By cracking the human genetic code, we are now able to benefit from better medical treatments.

How much code does it take to make programs run?

Look at this visual that compares the number of lines of codes used in different software: Million Lines of Code — Information is Beautiful

These figures are, at best, an estimation but it gives us an idea of the incredible range. Of note is the estimated 2 billion lines of code in the software that runs Google's Internet services, i.e. Google Maps, Gmail etc! That IS massive!

Do animals problem solve?

Animals have not only been observed to be able to solve problems – many, from sea otters to chimpanzees, have even used tools in the process.

There are macaques that use rocks to open oysters, octopuses that store away coconut shells for later use as shelter or armour, and even orangutans that have fashioned whistles from bundles of leaves to scare off predators! Even more surprisingly, New Caledonian crows have been observed MAKING tools by turning twigs into hooks to dig out food!

- Crows use hooked tools (cosmosmagazine.com)
- 10 Animals That Use Tools | Live Science

How can I make a video game?

It's a good question to ask because surely playing video games is part of the answer! Not merely playing, but also learning from games to understand what makes a game successful.

There are many different types of software that now make it easier for people to create video games. Game engines (e.g. Unreal, Unity, and GameMaker), are software 'templates' that can be used by game developers to build their worlds on, without having to code from scratch. There are also online communities of game developers who are happy to share their experience and artwork.

However, nothing replaces the need to have solid foundation in coding to be competent in using programming languages, such as Python and C#.

What are software bugs?

Software bugs are errors or flaws in computer programs. Bugs cause the software program to behave in an unexpected manner (e.g. no response, or incorrect response). Bugs could be brought about by the programmer not understanding the problem fully, not having enough time, or not having enough experience to know what pitfalls to avoid. Debugging is the process of correcting the errors. One thing is certain – these bugs are not related to bees or beetles.

One of the most famous bugs was the Y2K bug. In the 1900s, most software developers didn't think it was necessary to add in the digits 1 and 9 to represent each year. Because of that, 1998 was simply considered to be 98. But as the world drew closer to the year 2000, many people feared that systems everywhere would fail at the stroke of midnight, December 31 1999, because the computers all over the world would think that the following day would be Jan 1, 1900. A lot of money was spent to upgrade systems worldwide to ensure the world did not fall into chaos because of the bug. It didn't!

OUTSIDE OR SUPPLEMENTARY READING

Interactive tools and activities using block-based coding

- Storytelling overview - CS First
- Learn today, build a brighter tomorrow. | Code.org
- Pencil Code
- Scratch - Imagine, Program, Share (mit.edu)
- Blockly Games

Computational thinking

- What is computational thinking? - Introduction to computational thinking - KS3 Computer Science Revision - BBC Bitesize

TOPIC WORDS

- Algorithms
- Input
- Output
- Computational thinking
- Decisions
- Reasoning
- Logical
- Problem solving
- Code
- Instructions
- Programming

PRIMARY + STEM

For more teaching resources, visit

WWW.PRIMARYANDSTEM.ONLINE

Supported by

The Invergowrie Foundation

Swinburne University

The INVERGOWRIE Foundation

SWINBURNE UNIVERSITY OF TECHNOLOGY