Computational Thinking for Teachers
Computational Thinking for Teachers
The Commonwealth of Learning (COL) is an intergovernmental organisation created by Commonwealth Heads of Government to promote the development and sharing of open learning and distance education knowledge, resources, and technologies.

© 2021 by Commonwealth of Learning

Computational Thinking for Teachers is made available under a Creative Commons Attribution-ShareAlike 4.0 Licence (international):
http://creativecommons.org/licences/by-sa/4.0.

For the avoidance of doubt, by applying this licence the Commonwealth of Learning does not waive any privileges or immunities from claims that they may be entitled to assert, nor does the Commonwealth of Learning submit itself to the jurisdiction, courts, legal processes, or laws of any jurisdiction.

Authors:
Kaushal Kumar Bhagat, Indian Institute of Technology Kharagpur, India
Chandan Dasgupta, Indian Institute of Technology Bombay, India

Acknowledgements
This course module would not have been possible without the valuable assistance of Shalini Toluchuri, who is doing her Ph.D. at the Indian Institute of Technology Kharagpur, India. We would also like to thank Dr Sanjaya Mishra, Education Specialist: eLearning, Commonwealth of Learning, for his valuable suggestions to improve this course.

Published by:
Commonwealth of Learning
4710 Kingsway, Suite 2500
Burnaby, British Columbia
Canada V5H 4M2
Telephone: +1 604 775 8200
Fax: +1 604 775 8210
Web: www.col.org
Email: info@col.org
# Table of Contents

**About the Course** 5

**Lesson 1: Introduction to Computational Thinking** 6
- Introduction 7
- Definition of computational thinking 9
- Different components of computational thinking 11
- Misconceptions about computational thinking 31
- Test your knowledge 32

**Lesson 2: Data Practices** 35
- Introduction 36
- Data gathering/generation 38
- Understanding qualitative data generation and quantitative data generation 39
- Data-generation tools 40
- Data manipulation 43
- Data analysis 48
# Computational Thinking for Teachers

## Data visualisation

[50]

## Test your knowledge

[52]

## Lesson 3: Computational Problem Solving

### Introduction

[56]

### How can you solve problems computationally?

[57]

### Understanding the problem

[59]

### Pattern recognition

[66]

### Creating abstraction

[71]

### Algorithms

[74]

## Test your knowledge

[81]

## Lesson 4: Modelling and Systems Thinking

### Introduction

[86]

### What are computational models?

[87]

### Design, construct, evaluate and use computational models

[88]

### Computational modelling and systems thinking

[96]

## Test your knowledge

[101]

## References

[104]

## Image Attributions

[107]

## Answers to the “test your knowledge” questions

[108]
Computational thinking (CT) can help to enhance the problem-solving skills required to solve complex real-world problems. It can be applied across all disciplines, irrespective of students’ grade level. However, the primary goal of this course is to help teachers/educators understand CT and implement it in schools.

**Target Audience:** The course can be used by anyone with higher secondary qualification. However, the primary users will be students and teachers.

**Prerequisites:** There are no prerequisites for this course. No programming skills are required.

**Course Objectives:** Upon completion of the course, learners should be able to:
- define and explain computational thinking
- differentiate computational thinking from computer science and mathematical thinking
- recognise algorithms as important tools for problem solving
- describe the skills involved in computational thinking
- apply computational thinking in different disciplines
- examine existing curricula in light of computational thinking practices
- combine various computational thinking practices to redesign their classroom experiences

**Duration:** The course is developed to help learners use it flexibly and devote time according to their convenience. It is expected that on average, you will need about 120 minutes to complete this course.
Lesson 1

Introduction to Computational Thinking
Computational thinking (CT) recognises computational aspects of the world around us and the techniques to perceive and understand natural processes. CT has the potential to make changes in the rapidly developing world and deliver significant economic benefits for individuals and society. Accelerating technological advancements and monumental social demands push one to rethink the core fundamentals of CT for scientific approaches. CT is a collection of problem-solving approaches that involve expressing problems and resolving them in a way that even a computer can also execute. This is one of the fundamental abilities required for everyone to meet the demands of the Fourth Industrial Revolution (or Industry 4.0). CT skills include the mental ability set that transforms complex, confused and partially identified real-world issues into a type that can be solved even by a mindless machine, without additional human assistance. Computational reasoning can be used to build a modern prototype for thinking about and interpreting the world more closely. Hence, CT is relevant for learning beyond academics.
Learning outcomes

On completion of this self-study lesson, you should be able to:

• define the concept of CT
• identify different components of computational thinking
• distinguish between CT and computer science
• apply CT in different domains in school curricula
Computational Thinking for Teachers

**Definition of computational thinking**

“Computational thinking involves solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science” (Wing, 2006, p. 33).

Although, there is no clear definition of CT, it can be described as a strategy that implies a structured approach to problem solving. CT includes countless abilities and procedures to achieve a desired solution. With the advancement of CT, we will be able to strengthen computer science applications and solve problems in a variety of domains, including maths, humanities, science, and the arts.

CT requires the following skills:

- Building problems in a way that allows us to use software and other tools to help solve them.
- Evaluating and arranging data logically.
- Representing data using models, simulations and graphs.
- Automating solutions into an algorithmic pattern with a sequence of ordered steps.
- Achieving, defining, evaluating and executing potential solutions using the most organised and effective combination of steps and resources.
Classroom activity

Ask the students to write a recipe for making Maggi noodles, carefully considering every step required. Remember that these step-by-step instructions comprise an algorithm. Read the different recipes, then address the levels of simplification (abstraction) present in them. Ask the students to share each other’s recipes and check whether they observe any similar patterns.

Ask older students to work on a more challenging project individually or collaboratively. For example, they can research and write out aspects of a curriculum topic such as the impact of social movements on culture and legislation, or they can plan a meeting or a classroom play. In each case, ask them to document the steps required to perform the task, then focus on what they have left out of their report.
There are four key fundamental techniques for engaging in computational thinking, shown in Figure “The four cornerstones of computational thinking.”

1. Decomposition
Breaking down a complex problem or structure into smaller, more manageable elements.

2. Pattern recognition
Looking for similarities within problems.

3. Abstraction
Focusing only on the important information, ignoring irrelevant details.

4. Algorithms
Developing a step-by-step solution to the problem, or the rules to solve the problem.
Decomposition

Decomposition is the process of breaking down a complex problem into much smaller and more feasible pieces of. Big problems can be overwhelming, so it is often simpler to operate with a set of smaller, related tasks, making the procedure a manageable and realistic one. The sections of the decomposed problem can then be separately formed, understood, evaluated and solved.

This helps to address complicated issues quickly and makes it easier to design the implementation of large systems. For example, breakfast can be broken down into individual tasks (see Figure “Breaking down breakfast into a set of steps”), such as toasting, making tea or coffee, and boiling an egg. In turn, each of these can be broken down into a series of different steps as well. Each component can be created and adapted later in the process through the decomposition of the original task.

Examples of decomposition in the curriculum

English language

When considering a story or a subject, analyse its concepts by answering: Who are the protagonists and antagonists? What is the content’s main idea? What is the conflict? How is the conflict resolved? List the answers separately.
**Mathematics**

As shown in Figure “Decomposing individual components of a quadratic equation”, we can use decomposition in solving quadratic equations to find the value of an unknown variable.

**Science**

The water cycle can be explained through a series of steps, from evaporation to precipitation. Figure “Processes in the water cycle” shows a labelled water-cycle diagram, which can be decomposed into different processes, such as evaporation, condensation, precipitation, transpiration and percolation.

**Decomposing individual components of a quadratic equation**

**Solve** $X^2 – 8X + 12 = 0$

**Step 1**: $X^2 – 2X – 6X + 12 = 0$

**Step 2**: $X (X – 2) – 6 (X – 2) = 0$

**Step 3**: $(X – 2) (X – 6) = 0$

**Step 4**: $X = 2, 6$
Social Studies

By researching various populations’ customs, practices and history, students can discover and learn about different cultures, integrating decomposition into the social sciences.

Arts

Many tasks must be performed to create a short film, including writing a story, casting the actors, filming and editing.

Languages

Sentence structures can be learned by breaking sentences into various parts or grammatical units, such as subjects, objects and verbs (see Figure “Breaking a sentence into grammatical units”).
Classroom activity

Ask the students to plan for a Halloween party. For example, the process can be decomposed into deciding what food to have, recruiting others to help with cooking, selecting decorations, organizing music, deciding what time to have the party, and sending out invitations.

Decomposition will come into play from a computer science and coding perspective if students decide to design a new game — Ludo, for example. Students should analyse the characters, the setting and the story, and see how different strategies can be applied.
Pattern recognition

Pattern recognition includes discovering similarities or patterns in any complex or decomposed problem. Identifying patterns can help us resolve a problem more effectively. By detecting similarities between the patterns within problems, we can make predictions, create rules and solve the problems. Figure “Identifying similar patterns” depicts a smartphone game in which three green diamond formations are identical. Students can be asked to compare similar objects and find commonalities between them. Young students may develop an understanding of patterns by determining what the objects have in common.

The identification of patterns is the recognition of common features between ideas and objects. Pattern recognition is a method of data reduction whereby data are grouped according to visual or logical patterns, based on their characteristics and relationships. Algorithms can be adapted to solve a whole class of similar problems, yielding a general solution that can be implemented if a similar type of problem is subsequently encountered.
Examples of pattern recognition in curricula

English language

Students can strengthen their vocabulary acquisition by grouping words that have similar synonyms or similar etymological roots.

Mathematics

The fundamental trigonometric functions are used to measure slopes and intercepts, which can be adapted from the Pythagorean theorem, as shown in Figure “Pythagorean theorem”. Another example is the principle of mathematical induction, whereby students learn to solve a problem in a specific sequence.
Science

<table>
<thead>
<tr>
<th>LIVING</th>
<th>NON-LIVING</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Characteristics:</strong></td>
<td><strong>Characteristics:</strong></td>
</tr>
<tr>
<td>• Can grow</td>
<td>• Cannot grow</td>
</tr>
<tr>
<td>• Can move</td>
<td>• Cannot move</td>
</tr>
<tr>
<td>• Can breathe</td>
<td>• Cannot breathe</td>
</tr>
<tr>
<td>• Can eat</td>
<td>• Cannot eat</td>
</tr>
<tr>
<td>• Have senses</td>
<td>• Do not have senses</td>
</tr>
<tr>
<td>• Can reproduce</td>
<td>• Cannot reproduce</td>
</tr>
</tbody>
</table>

**Examples:**
- Mammals, other animals, humans, fruits, trees, flowers
- Machines, vehicles, books, smartphones, ball, table, chair

**Classifying living and non-living things according to traits**

Social Studies

Students can identify the possible effects of various economic developments by looking at data patterns to assess interest rates and inflation.

Based on characteristics, identify living and non-living things and describe similar traits for grouping them (see Figure “Classifying living and non-living things according to traits”).
Arts

Students can describe sonnets or other poems based on similarities and identify words that rhyme.

Classroom activity

Help students create rhythmic musical compositions using simple sequencing software that repeats patterns of beats.

Give students a pattern-identification game requiring them to align a tile with its identical counterpart. The equivalent tile can be rotated to make this a more challenging brain-training activity. See https://www.braingymmer.com/en.brain-games/pattern_matrix/play/
Before proceeding further, watch the video on “Abstraction”.

**Watch Video:** [https://www.youtube.com/watch?v=WpwE972jV_8](https://www.youtube.com/watch?v=WpwE972jV_8)

Video attribution: “Abstraction” by Commonwealth of Learning is available under CC BY-SA license.

**Abstraction**

“The abstraction process — deciding what details we need to highlight and what details we can ignore — underlies computational thinking” (Wing, 2008, p. 3718).
The main aim in CT is to concentrate only on the important data and disregard irrelevant information. Abstraction makes it easier to solve issues or problems by removing unnecessary details. It is the process of creating an artifact that is more understandable. Abstraction may be the most complex step of CT, but it helps with filtering out — minimising — the attributes of patterns that don’t require our attention, helping us create a picture of what we are trying to solve.

A good illustration of abstraction is making coffee with a coffee machine, as shown in Figure “Abstraction of a coffee machine”. To make coffee, one needs to know how to use a coffee machine. We need to choose the type of coffee we want to drink and the size, then turn on the machine. How the coffee machine works internally to brew a fresh cup of delicious coffee is not something we need to know. We don’t need to consider the best water temperature or the number of coffee beans required, as the coffee machine takes care of these details. We only communicate with a primary interface that requires minimal understanding of how the coffee machine works.

**Examples of abstraction in curricula**

**English language**

Students can summarise a novel into a short review consisting of only the main ideas.
For any scientific experiment carried out in the curriculum, students may summarise their understandings and final results. For instance, in Figure “Abstract representation of Newton’s laws of motion”, Newton’s laws of motion are represented pictorially. Students should be able to learn new laws and theorems to solve other scientific problems.

**Mathematics**

Students can perform a class survey of final examination results and then analyse the data to record the primary outcomes, produce a table and present the results.

**Science**

![Abstract representation of Newton's laws of motion](image)
Social Studies

Students can extract the most relevant details about a particular event, such as a book launch party, and use them to write a concise report about it.

Arts

Students can categorise chord progressions into a set of general principles for composing in popular musical genres (see Figure “Categorising chords to generate music”).

Categorising chords to generate music
Classroom activity

Ask the students to create games for people who are beginners at coding computer programs. If the games are based on real-world structures, then the difficulty can be handled using abstraction. They can design one-minute games such as Cup Stack, in which players must stack paper cups on a table to make a pyramid.

Ask the students to form groups and discuss real-life examples where we use abstraction. For instance, using an ATM machine is a clear example of abstraction. We don’t know the internal workings of an ATM, but we know it allows us to perform operations like cash withdrawal, money transfer, mini-statement retrieval, etc. Another example is riding a motorcycle; we know how to do this, but we don’t necessarily know the inner workings of the vehicle itself.
An algorithm is a finite set of well-defined instructions that a computer can carry out in sequence, usually to solve general issues or perform a computation. Algorithmic reasoning using a concise overview of steps is a way of reaching the solution to a problem. When similar issues have to be solved over and over again, algorithmic thinking comes into the frame.

Once an algorithm is known, it does not have to be worked out from scratch for a new but related problem. To do multiplication or division at school, an algorithm can be implemented. The solution to any multiplication equation can be found if simple rules are followed precisely by software or humans.
Figure “Algorithm designed to check whether a lamp is broken” presents an algorithm built to check whether a lamp is broken. Figure “Algorithmic process to prepare tea” presents one for making tea.
Examples of algorithms in curricula

English language

<table>
<thead>
<tr>
<th>Present Perfect Tense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subject</strong></td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td>You</td>
</tr>
<tr>
<td>They</td>
</tr>
<tr>
<td>We</td>
</tr>
<tr>
<td>She</td>
</tr>
<tr>
<td>He</td>
</tr>
<tr>
<td>It</td>
</tr>
</tbody>
</table>

Table for understanding the present perfect tense

Students can use algorithms to learn about important aspects of English grammar, such as the present perfect tense or past perfect tense. As shown in Figure “Table for understanding the present perfect tense”, the student can use a flowchart or table to improve their understanding of grammar and hence their everyday speaking and writing skills.
The addition of two numbers can be broken down into sequential steps, as shown in Figure “Algorithm for the addition of two numbers”, and this technique can be used for adding $n$ numbers.
Science

Students can classify elements in the periodic table to learn the properties of each element.

Social Studies

Students can describe historical events sequentially.

Arts

Students can draw a portrait by following a sequence of steps. These steps might involve an algorithm that can be used by other students to draw their own images.
Classroom activity

Ask the students to form groups and solve the 2048 numerical puzzle. In this puzzle, if two tiles meet, they combine to give you a larger tile that shows their number. The faster you do this, the higher the tiles get and the more crowded the board gets. Your aim is to hit 2048 before the board is completed. There can be several variations in the puzzle. This offers a good way to refine students’ understanding of using algorithms to solve problems. See https://www.youtube.com/watch?v=CcrWCdsPeMc.

Ask the students to sort a set of unknown weights by means of a standard pan balance. Closely examine the algorithm they follow to do so, and then think of a faster way to do the same job. See https://www.advancedict.info/interactive/algorithms.html.
Most teachers assume that CT is about problem-solving using computer technology. They think CT has something to do with creating new algorithms and using a programming language to implement them. But CT isn’t just about computer science or programming; it’s about visualising and solving a complex problem efficiently and effectively.

Facts about computational thinking:

- **Misconceptions about computational thinking**
  - It is not about writing in programming code; it’s about fully comprehending a problem, organising it logically and analysing the relevant details.
  - It is not about rote skills; it’s about grasping the basics and applying them to a generalised range of problems.
  - It is about thinking like computer scientists, not like computers.

<table>
<thead>
<tr>
<th><strong>Computational Thinking</strong></th>
<th><strong>Computer Science</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Split a problem into components or measures.</td>
<td>Computer science is a multifaceted field that includes mathematical and engineering elements, which manifest together as algorithmic problem-solving processes.</td>
</tr>
<tr>
<td>Recognise and locate trends or patterns.</td>
<td>Visualise data comparing the material of a microprocessor and the corresponding computer speed to observe whether there is a trend.</td>
</tr>
<tr>
<td>Develop an instructive approach for solving a problem, or steps for a task.</td>
<td>Write a computer program to sort data.</td>
</tr>
<tr>
<td>Generalise trends and patterns through laws, beliefs or observations.</td>
<td>Apply data structures to obtain and use information in a program more efficiently and effectively.</td>
</tr>
</tbody>
</table>
1. Decomposition is
   b. Observing patterns, trends, similarities and differences across data.
   c. Identifying the general principles that generate these patterns.
   d. Breaking down data, processes or problems into smaller, manageable parts.

2. Pattern recognition is
   a. Identifying the general principles that generate patterns.
   b. Observing patterns, trends, similarities and differences across data.
   c. Developing the step-by-step instructions for solving this and similar problems.
   d. Breaking down data, processes or problems into smaller, manageable parts.
Computational Thinking for Teachers

3. Abstraction is
   a. Observing patterns, trends, similarities and differences across data.
   b. Breaking down data, processes or problems into smaller, manageable parts.
   c. Identifying the general principles that generate patterns.
   d. Developing the step-by-step instructions for solving this and similar problems.

4. What method would you use to find a common theme across multiple events that led to the American Revolution?
   a. Abstraction
   b. Algorithms
   c. Pattern recognition
   d. Decomposition

5. Algorithm design means:
   a. Identifying the general principles that generate patterns.
   b. Developing the step-by-step instructions for solving this and similar problems.
   c. Observing patterns, trends, similarities and differences across data.
   d. Breaking down data, processes or problems into smaller, manageable parts.

6. Designing a science experiment based on a famous scientist’s previous experimental design requires what activity?
   a. Abstraction
   b. Algorithm
   c. Pattern recognition
   d. Decomposition
7. Which one of the following is an example of computational thinking?
   a. Allowing your friend to decide where to go.
   b. Asking a parent to arrange for you to meet your friend.
   c. Planning when and how you will meet a friend.
   d. Wandering around until you find a friend.

8. Why do we need to think computationally?
   a. To help us learn programming.
   b. To address complex issues more easily.
   c. To make us think like a computer.
   d. To create new lessons for the school curriculum.

9. Pretend you created an algorithm to put these last names in alphabetical order: Jamal, Berthe, Sulye, Wyatt. After an algorithm ran the first time, what would the order of the names be?
   a. Berthe, Jamal, Sulye, Wyatt
   b. Wyatt, Sulye, Berthe, Jamal
   c. Jamal, Berthe, Wyatt, Sulye
   d. Sulye, Berthe, Wyatt, Jamal

10. What is computational thinking?
    a. It addresses a big or complex problem by breaking it down into smaller pieces.
    b. It makes us think like a robot.
    c. It makes us memorise steps.
    d. The solution to a problem lies in developing software.
Lesson 2

Data Practices
Data practice skills augment the way information is represented and improve the use of information for STEM subjects (science, technology, engineering, and mathematics). Data practices include the collection, manipulation, analysis and visualisation of data in science and engineering. Teachers should consider these activities as necessary procedures that students must follow to obtain accurate and reliable data and draw robust conclusions about scientific phenomena. Figure “Data practices in computational thinking” shows the sequential order of data transformations that intensifies the quality of the data and can deliver the ideal solution.
Learning outcomes

On completion of this self-study lesson, you should be able to:

• identify the merits of data practices in STEM
• understand qualitative and quantitative data
• apply different data-generation tools
• identify the merits of data manipulation
Data gathering or generation requires qualitative research and is not a method of automatic mapping. Data-generation management involves the production of data from experiments or observation. Figure “Data can come from many different domains” shows that data can be obtained from different sectors, such as health care, modern electronic media, e-commerce, and many others. Interviews, findings, questionnaires, and records can be used for data collection. Quantitative data refers to anything that can be counted. Questionnaires, surveys, papers and reports are typically quantitative, whereas interviews, focus groups and observations are qualitative.
Qualitative data are non-numerical in nature and can be collected through observations, face-to-face interviews, focus groups, record keeping, case studies, etc. For example, if a student recites a poem, and the teacher gives feedback based on the student’s fluency, word pronunciation, and rhythm instead of assigning a numerical score, this feedback is qualitative data. Quantitative data refers to evidence based on numbers. It is more objective in nature and can be collected through questionnaires and surveys. The collected data can then be analysed statistically. For example, imagine a teacher is teaching the same course to two different groups of students. At the end of both courses, the teacher wants to determine which group has performed better. He administers a test and assigns scores based on the students’ performance. The scores assigned by the teacher are quantitative data that will help him find out whether there is a significant difference between the two groups’ performance.
Interviews

Interviews are a straightforward way to collect information on an issue. Almost anyone can develop an inventory of questions, but knowing what to ask is the trick to successful interviews. In most cases, the interviewer is a subject expert who wants to understand others’ opinions and views by asking well-planned and well-managed questions.

Observation

Observation refers to a method of collecting data through observing. The researcher notes down the behaviour and activities of the participants in a scenario, for example.
Documents and records

Analysis based on documentation and records often uses current data for a review. Reports of attendance, minutes of meetings, and financial records are just a few examples of this kind of research. It is also effective and economical to use documents and records, as these data are readily available.

Focus groups

Focus groups are another form of qualitative data collection. The data are more descriptive in nature and cannot be measured numerically. This method involves a group of participants (usually five to ten persons) who share similar interests. For example, a selected group of participants discusses their opinion on a particular topic, and the researcher gathers information from participants’ individual views.
Questionnaires and surveys

A survey is a common method for gathering and analysing statistical data. Survey questions are generally closed-ended. On the other hand, a questionnaire is an instrument that consists of both open-ended and closed-ended questions. This provides a bigger picture than a survey.

Classroom activity

Ask the students to perform a study in which they gather qualitative and quantitative information to address the question: "Has global warming changed the quality of life?"
Data manipulation is the process of altering and modifying data to make it simpler and more meaningful to interpret. The manipulation of data is an integral part of CT. Data must be designed so it can be read by computers. For humans to use it and machines to comprehend it, however, data requires prior manipulation. Since the amount of information used and stored is increasing every day, data manipulation has become more necessary than ever before.

Data manipulation is about changing data to make it easier to read or organise. As shown in Figure “Converting raw data into an organised form”, we use manipulation to bring organisation to unstructured data. For example, a school must maintain a log of student data for academic purposes. We can use data manipulation by sorting or rearranging the logs in alphabetical order to make it easier to identify individual entries. Data modification makes details more organised and easier to understand. For online searches, data manipulation is generally used to allow a researcher to view the most relevant web pages.
Purpose of data manipulation

Data consistency

Data that is consistent can be interpreted and understood very easily. There may not be a unified view of data from various sources. However, data manipulation makes it simple to organise and store data.

Data projection

Data manipulation makes it possible to use historical data, especially when it includes financial transactions, to provide in-depth analysis.
**Building data value**

Data manipulation allows the addition, deletion, insertion or transformation of information. Data that is static is no longer useful. But with sufficient data-handling skills, data can be produced to make smarter decisions.

**Elimination of redundancy and irrelevancy**

Data manipulation also helps to get rid of redundant or irrelevant information. Data sorting, filtering and cleaning techniques exist to help us do this. Re-sorting is a method that aims to adjust the order of data. Re-sorting is most frequently applied to quantitative data, but it can also be widely implemented with text-based material. There are a few common sorting methods, such as numerical, alphabetical and chronological. Sorting data helps separate important individual attributes, such as first or last, the very best or the lowest, the most frequent, and so forth.
Rearranging is another simple method for data manipulation, involving the repositioning of information. Examples include rearranging furniture, categorising a collection of images by sorting them into identical stacks, or organising files on a desktop.
Classroom activity

Ask students to create a “go green” plan for their school. Separate approaches may be planned, such as paper recycling, minimum usage of electricity and water, and composting food waste.

Ask students to create a publication plan for a monthly magazine by defining tasks, duties, timelines and resources required for the project’s completion.
Data analysis is a process of transforming and modelling data to obtain useful decision-making information. This process aims to extract valuable information from the data and make a choice based on the analysis. For example, in an online shopping platform, customer complaints are collected from chatbots, emails, phone calls, etc. Requests are converted into tickets that get rated on a scale of priority and are automatically routed to the right department or individual employee. This process reduces the customer’s wait time and ensures they receive expert guidance.
Classroom activity

Ask the students to attend a car racing event, then tell them to analyse the results based on car speed and the time taken to complete the race.

Use appropriate statistical methods that will best test this hypothesis: “The development of mobile devices has improved the quality of life.”
Data visualisation is the visual representation of data to convey information more clearly and effectively for humans. We can convey findings better by visualising data via representations such as graphs, charts, infographics and maps. Data visualisation helps with identifying underlying patterns and trends in data.

Let’s take a look at a few typical examples to show how data visualisation is related to our daily life.

**Fitness trackers**

Fitness trackers are wearable devices that provide daily health data such as distance walked or run, heartbeat, pulse rate and calorie consumption using infographics.

**Bus schedules**

Bus schedules are another form of data visualisation, using tables to present information.
Newspapers

Data visualisation is used in newspapers to make complex data more comprehensible and attractive — for example, weather reports, stock news, etc.

Classroom activity

Ask students to create a spreadsheet to simulate the “Birthday Problem” — i.e., how many people must be in a room for there to be at least a 50% chance that at least two have the same birthday. Use the same model to answer the question for three people having the same birthday.

Ask students to use an appropriate model of a simple ecosystem to conduct experiments that determine what will happen to the ecosystem if some percentage of the living creatures go extinct. How will these extinctions affect human growth?
1. Data generation is
   a. Making sense of data, finding patterns and drawing conclusions.
   b. The process of gathering appropriate information.
   c. Depicting and organising data in appropriate graphs, charts, words or images.
   d. Breaking down tasks into smaller, manageable parts.

2. Data-gathering methods include
   a. Transformation and recognising similar patterns
   b. Sorting and rearranging
   c. Flowcharts, graphs and images
   d. Quantitative and qualitative approaches
3. Quantitative data-collection tools include
   a. Focus groups
   b. Observations
   c. Interviews
   d. Surveys

4. Data manipulation is
   a. Breaking down tasks into smaller, manageable parts.
   b. Making sense of data, finding patterns and drawing conclusions.
   c. Depicting and organising data in appropriate graphs, charts, words or images.
   d. The process of gathering appropriate information.

5. Data manipulation methods include
   a. Transformation and recognising similar patterns
   b. Quantitative and qualitative approaches
   c. Flowcharts, graphs and images
   d. Sorting and rearranging

6. Categorising a set of photographs by grouping them into similar piles based on their dates is
   a. Data manipulation – rearranging
   b. Data analysis
   c. Data visualization
   d. Data collection
7. Data analysis is
   a. Breaking down tasks into smaller, manageable parts.
   b. The process of gathering appropriate information.
   c. Making sense of data, finding patterns and drawing conclusions.
   d. Depicting and organising data in appropriate graphs, charts, words or images.

8. Data analysis approaches include
   a. Sorting and rearranging.
   b. Flowcharts, graphs and images.
   c. Transformation and recognising similar patterns.
   d. Quantitative and qualitative approaches.

9. Data visualisation is
   a. Depicting and organising data in appropriate graphs, charts, words or images.
   b. The process of gathering appropriate information.
   c. Breaking down tasks into smaller, manageable parts.
   d. Making sense of data, finding patterns and drawing conclusions.

10. Data visualisation uses the following methods:
    a. Sorting and rearranging.
    b. Flowcharts, graphs and images.
    c. Quantitative and qualitative approaches.
    d. Transformation and recognising similar patterns.
Lesson 3

Computational Problem Solving
Introduction

Solving a problem using computational thinking often involves reframing the problem in ways that allow it to be solved in a systematic, step-by-step manner. It does not necessarily mean that computers have to be used to solve the problem. Instead, this method of problem-solving emphasises the need to prepare the problem and think about solutions in such a way that a human or machine can solve it by following the steps provided. There is no one right way of preparing problems for solving by computational methods.

Learning outcomes

On completion of this self-study lesson, you should be able to:

• explain the problem that needs to be solved
• break a complex problem into smaller, simpler sub-problems
• solve a problem using computational methods and strategies
• recognise and construct patterns useful for solving a problem or sub-problem
• generate abstractions pertaining to a given problem
• write algorithms and troubleshoot errors
How can you solve problems computationally?

In general, to solve a problem computationally, one needs to first understand the problem, then divide the problem into smaller and simpler sub-problems, find similarities or patterns, identify important sub-problems that need to be tackled first, start solving those problems, and reflect on your work to think of improvements (Dromey, 2008; Pólya, 1957). This process is applied repeatedly until the solution to the original large, complex problem is reached.

Let’s dig deeper into how to solve problems computationally, using an example problem that can also be used in class. Note that the example in the class activity suggested below is ideally suited for students aged five to seven but can be used as a starter problem or icebreaker in higher grades as well. Further, the problem can be made complex by adding further constraints, such as the shortest path, adding more obstacles, such as stopping to get groceries for Grandma or having to avoid multiple wolves along the way, or comparing different starting positions. A computer is not required for solving this problem.
Classroom activity

Rescue mission: You are in charge of helping Little Red Riding Hood rescue her grandmother from the big bad wolf. Write a program that will help her reach grandma’s house quickly and safely.
Understanding the problem

It is essential to accurately understand the problem one is being asked to solve. Skipping this step or rushing to the solution will likely lead to inefficient and ineffective solutions for the actual problem. Thus, the emphasis in this step is to focus on what is the problem. Note that here we do not worry about how to solve it. Below are some sample prompts that may help students and novices understand a problem accurately (Pólya, 1957). These prompts check for problem comprehension and encourage students to engage deeply with the problem statement. Please select, modify and create prompts to suit your students’ grade and learning requirements.

- Are you able to understand all the words used in the problem statement?
- Can you represent the problem using a picture or diagram?
- Can you list all the factors that are involved in this problem?
- Can you rephrase this problem in your own words?
- Can you think of a similar problem that you may have solved or come across in your life earlier?
- What information do you have that will help solve the problem?
- What additional information do you need to solve the problem? Can this be generated from the information that’s already given?
- Are there details in the problem statement that do not matter for the solution?
- What is (are) the most important piece(s) of information in the problem statement?
Classroom activity (continued...)*

Back to helping Little Red Riding Hood reach her grandma. Draw a diagram to represent the problem. For example, the diagram below can be drawn on the floor of the class or any other big surface.

* Adapted from CSUnplugged (n.d.).

In the classroom activity suggested above, the basic idea is that the original problem did not provide information about where grandma’s house was located. The diagram helps understand the problem in that regard. Also, we know that a wolf will get in the way and that Red Riding Hood needs to avoid the wolf. The location of this wolf can be added to the diagram as well.
Problem decomposition

After the student has begun the journey to understand the actual problem, they need to break the original problem into smaller parts—i.e., decompose the problem. This step is often called “divide and conquer” because it involves breaking down data, processes or complex problems into smaller, simpler, manageable parts. There may be pre-existing solutions to some of the sub-problems, which can then be used to think about the solution to the original, bigger problem.

The 17th-century philosopher René Descartes famously said: “Divide each difficulty into as many parts as is feasible and necessary to resolve it.”

To reiterate: any large, incomprehensible or challenging problem comprises multiple smaller sub-problems. Thus, the way to solve big problems is to break them into smaller parts and solve those sub-problems one by one. If you get stuck in one of the sub-problems, then break it down further into smaller sub-problems that can be solved easily. Coming up with possible solutions is much easier if the problem is well defined. Some strategies for decomposing problems are as follows.
1. Repeated questioning

Here, the goal is to iteratively move towards the root cause of the original problem by repeatedly asking questions such as why, what, where, how, when, etc., thereby uncovering smaller sub-problems.

For example, in the Red Riding Hood example, questions such as “How far can she go every time?” “What directions can she go in?” and so forth can help uncover smaller sub-problems.

2. Functional decomposition

This refers to dividing a problem by thinking about its purpose or the purpose of the system that lies at the core of the problem. The goal is then to identify smaller components of the system, each having its own purpose and all working together to fulfil the system’s overall purpose.

a. Identify main functions and list them separately.

b. Prioritise them based on importance — i.e., determine which function, if missing, will render the solution useless.

c. Solve each independently.
For example, the primary directions required for Red Riding Hood are forward, turn left, and turn right. Thus, the bigger problem of indicating directions has now been decomposed into a set of smaller sub-problems concerning whether Red Riding Hood should move forward, turn left, or turn right. Forward and one of the turns (e.g., left) have a higher priority because she cannot reach Grandma’s house without these.

The second direction (e.g., right) has a lower priority, since she can turn right by taking multiple left turns; however, that greatly increases the number of steps, meaning the sequence will be inefficient, as it will take much longer to implement.

Other directions are also possible such as backward, diagonally or U-turn, but these have a lesser priority, since the house can be reached without moving in these directions. However, having these may further reduce the number of steps and thus can be considered in future.

**3. Division based on structure–behaviour–function**

Here, the problem or the system that forms the core of the problem is divided into the underlying structure (i.e., elements of a system), behaviour (i.e., how the structures of a system achieve their purpose) and function (i.e., why an element exists within a given system — in other words, the purpose of an element in a system).

For example, structure helps us focus on the problem in terms of the constituent elements, such as Little Red Riding Hood, grandmother, wolf, etc. Behaviour helps focus on how these elements interact with each other to achieve their purpose, such as Little Red Riding Hood avoiding the wolf to reach Grandma’s house because the wolf can harm her. Finally, function helps focus on the overall purpose of the elements, such as that Little Red Riding Hood’s purpose is to move in the direction of her grandma’s house and rescue her.
Another example of the structure–behaviour–function type of decomposition is cleaning an aquarium (Hmelo-Silver & Pfeffer, 2004). Here, structure includes the fishes, plants and filter, since these are some of the elements that comprise an aquarium. These structural elements behave in a certain way. For example, the filters trap large dirt particles and absorb chemicals in the aquarium. Finally, the function of the filters is to keep the aquarium clean.

In our everyday lives, we don’t often give instructions as specific as, “Turn right, take a step forward, take another step forward, then turn right again”; it is much more straightforward to simply say, “Please go over there.” But when we program, we have to be very specific because we have to tell computers exactly how to do each thing and limit ourselves to the few instructions that they can follow. Students need to work on taking something they are familiar with, such as “go to that square,” and identifying the simplest individual steps that need to happen for someone to accomplish this.

Also, programs can be written incrementally; hence, instead of trying to solve the whole problem, students were encouraged to write a few steps first, test these, and then add to them. Breaking a program down into smaller components makes the task less overwhelming.

**Classroom activity (continued...)**

Teacher: “Bot, please pick up Little Red Riding Hood and be ready to receive the instructions from the teachers.” (The bot can carry a toy or token representing Little Red Riding Hood, or students can imagine that they are guiding her.)

The teacher then reads off the board: “Move forward, move forward.”
Problem decomposition is an important step towards solving a problem computationally. The following video helps better understand how to break down, simplify or decompose a complex problem.

Watch Video: https://www.youtube.com/watch?v=rWchjpeunbo

Video attribution: “Computational Problem Solving” by Commonwealth of Learning is available under CC BY-SA license.
“Patterns exist everywhere — in regularly occurring shapes or structures and in repeating events and relationships. For example, patterns are discernible in the symmetry of flowers and snowflakes, the cycling of the seasons, and the repeated base pairs of DNA” (National Research Council, 2012, p. 85). Patterns are essentially comprised of regular repeating relationships between elements. Generally speaking, there are three kinds of patterns (Papic et al., 2011):

1. Shapes with regular features (e.g., triangles with equal sides and angles, and shapes made with equally spaced dots, like on the sides of dice — see Figure “Three sides of a dice”).

2. Repeated sequence (e.g., AB sequences, like a red-blue, red-blue pattern; ABB patterns with repeating units, like red, blue, blue; see Figure “Repeating coloured blocks”).

3. Growing pattern (e.g., staircase with equal steps; see Figure “Growing staircase pattern”).
A series of patterns can be used to create beautiful art forms (see Figures “Warli art” and “Painting by Australian aboriginals”).

(Source: "A Warli painting," by Jivya Soma Mashe, "Wikipedia, is licensed under CC BY 2.0.

(Source: "Aboriginal Art Aboriginal Painting" from Pixabay, licensed under Pixabay licence.)
Pattern identification and construction techniques

Let’s now discuss some techniques that can help students learn pattern identification and construction for developing computational thinking.

1. **Copying and continuing a pattern**

Here, students can mimic a pattern created by their teacher or facilitator. They then continue building the same pattern on their own while preserving the original sequence being mimicked (e.g., picking up two red-blue coloured blocks at the same time). One can ask students to verbally state the sequence they are mimicking, thereby drawing their attention to the repeating pattern.

2. **Making or drawing patterns from memory**

You can show a pattern for some time, discuss the logic behind the pattern, come up with a heuristic (i.e., mental shortcut or rule-of-thumb for drawing the pattern), and then hide the original pattern. Students then need to reconstruct and draw the pattern from memory. Note that the goal here is not to memorise the pattern by rote but rather to understand the logic and generate a heuristic that will form the basis of the student-recreated pattern.

3. **Spotting a mistake and repairing the pattern**

Here, you can ask students to critically analyse a pattern (constructed by themselves or someone else) and identify mistakes in it. They can then attempt to repair the pattern.

4. **Explaining the pattern**

You can have students articulate in their own words what the pattern is. Additional prompts can help them move from a vague explanation (e.g., it’s a big pattern, or it’s a beautiful pattern) to a clear, generalised explanation (e.g., it’s like ABAB).
5. **Making your own pattern with similar or different objects**

You can have students translate the pattern to other modalities (e.g., with sound) or with different constraints (e.g., making the pattern with different objects but matching colours).

**Classroom activity (continued...)**

Teacher: “Bot, please pick up Little Red Riding Hood and be ready to receive the instructions from the teacher.” (The bot can carry a toy or token representing Little Red Riding Hood, or students can imagine that they are guiding her.)

The teacher then reads off the board:

1. move forward
2. move forward
3. move forward
4. turn left
5. move forward

* Adapted from CSUnplugged (n.d.).
Did we successfully program Little Red Riding Hood to rescue Grandma? How do we know?

*We know that we successfully programmed Little Red Riding Hood to rescue Grandma because Little Red Riding Hood and Grandma are in the same square.*

Can you think of other ways of reaching Grandma’s house?

*Many different patterns are likely to emerge when students write their programs. When students recognise these patterns, they can reuse some of them multiple times, rather than having to figure out each step again. For example, students may recognise that if they want Little Red Riding Hood to turn around 180°, they can repeat “left” or “right” twice, and they can repeat this same sequence whenever they need to do this movement. Or, if they need to move diagonally across the grid, they can take the group of instructions “left, go forward, right, go forward” (right and left swapped, depending on which way they want Little Red Riding Hood to go) and repeat this as many times as they need to cross the board.*

Now, can you identify some patterns, based on her movement and direction?

*What if we want Little Red Riding Hood to get to Grandma’s and then bring her back to safety?*
Creating abstraction

Abstraction is important, since it helps us focus on the important parts of a problem and solve it without getting distracted by less important information. The patterns that one has identified in a previous step usually form the basis for creating abstractions, since the repeating units indicate what are the important information pieces.

In the Little Red Riding Hood example, we have abstracted writing a program and using a programming language to very basic instructions of move forward, turn left or turn right. Students write these instructions down in familiar words or symbols and give the instructions to the bot verbally, which removes the need to know how to use a programming language and implement this on a computer. This helps students understand how sequences work in programming, without being overwhelmed by technical terminology and tools.

Consider another scenario, where one has to describe what cats look like. Information such as the colour of eyes, the length of tails and fur, the colour of fur, etc. are specific details that vary from one cat to another. However, most cats have a tail, four legs, two eyes, two ears, and fur. This is abstract information about cats that is true or accurate for a large majority cats. Thus, abstraction has helped us create a “model” of a cat. By looking at the model, we can identify that it represents a cat.
Techniques for creating abstraction

A model is a simplified or abstract version of any complex thing or living being. Here is one set of steps to create an abstraction or a model.

1. Identify repeating patterns (e.g., cat A has a yellow tail, cat B has a brown tail, cat C has a black tail, and so on; thus, most cats have a tail).
2. Categorise these and name the categories (e.g., tails of different colours can be categorised into one group called “tail”).
3. Repeat this for all the components that comprise the thing (e.g., all body parts of the cat).

Classroom activity (continued…)*

Add barriers to the grid so that the path is more complex because Red Riding Hood needs to avoid the barriers. This could be the Big Bad Wolf and other animals, or you could invent a new scenario for the grid. Ask students to write basic instructions of move forward, turn left or turn right. Students write these instructions down using familiar words or symbols and give the instructions to the bot verbally.
* Adapted from CSUnplugged (n.d.).
Algorithms refer to an ordered sequential set of instructions for solving similar problems or completing a task.

Some suggestions for how to write algorithms are provided below.

1. Begin by writing a high-level sequence of steps containing little detail. These steps can be informed by the abstract model developed in the previous step. For example, in order to bake a vanilla cake, a relevant algorithm could be:
   a. Gather all-purpose flour, sugar, butter, eggs, baking powder, milk, and vanilla extract.
   b. Mix these ingredients.
   c. Put in a heated oven.
   d. Remove the cake when it is done.

   These four steps in the high-level algorithm essentially bring together solutions of easy sub-problems underlying the bigger problem of how to bake a cake — what ingredients are needed, what to do with those ingredients, where to bake the cake, and what to do when it is done.
2. Now, do stepwise refinement, where further details are added to the high-level algorithm. To add these details, consider every step in the high-level algorithm, and add specific characteristics such as quantity, colour, duration, physical state, quality, temperature, etc. For example, a refined algorithm for baking a vanilla cake may look like the following:
   a. Preheat oven to 175°C.
   b. Mix the sugar and butter, followed by two eggs, beating them one at a time. Add a couple of drops of vanilla extract. Add flour and baking powder and mix well.
   c. Spread the mixture in a greased pan.
   d. Bake for 30–40 minutes in a preheated oven.
   e. Cake is done when it is spongy.

Note

This is now a recipe. All recipes are essentially algorithms giving detailed step-by-step directions for how to solve the problem of making some kind of food.
3. Now, refine the algorithm further by adding iterative components. It is seldom the case that a problem gets solved in the first attempt. If the problem is not solved yet, then we likely need to return to a previous step, repeating some of the existing steps and knowing when to stop this repetition. For example, if after baking for 30–40 minutes, the cake is not spongy yet, then the following can be done:
   a. If the cake is not spongy, then continue baking in the preheated oven.
   b. Check every 5 minutes to see whether the cake is spongy. Repeat step (a) until cake is done.
   c. Taste the cake to see whether it tastes like a good vanilla cake. In other words, has the problem of baking a vanilla cake been solved? If yes, then baking is complete. If not and the cake tastes like a chocolate cake, then restart by repeating step #2 with the correct set of ingredients.
Let’s go back to our Little Red Riding Hood example. Creating a sequence of instructions for this problem involves algorithmic problem solving, as it requires students to create an algorithm to accomplish a task. Computational algorithms are based on input, output, storage, sequence, selection and iteration. The following exercise focuses on sequencing instructions.

**Note**

*If* and *then* allow the baker to add conditions that specify how long to loop through the steps. Also, step (c) here pertains to troubleshooting when there’s a problem with the proposed solution and one is trying to figure out how to resolve the problem.

Algorithms allow other people or a computer to replicate a process by specifying the exact sequence of steps that need to be followed.

Let’s go back to our Little Red Riding Hood example. Creating a sequence of instructions for this problem involves algorithmic problem solving, as it requires students to create an algorithm to accomplish a task. Computational algorithms are based on input, output, storage, sequence, selection and iteration. The following exercise focuses on sequencing instructions.

**Classroom activity**

Have the students revisit the abstract instructions generated in the previous section, and tell them to refine these instructions by adding details and iterative components, as mentioned in this section. Students can create a flowchart or make a numbered list of steps indicating which steps follow each other, how many times a step should be repeated, etc.
**Troubleshooting**

Once the algorithm has been created, it is important to determine whether the sequence of steps outlined in the algorithm is doing what it is supposed to do — i.e., behaving correctly according to the specifications given in the problem. Troubleshooting refers to this investigation, where the purpose is to figure out whether everything is working properly and to determine why something is not working or behaving as expected, if that’s the case (Weintrop et al., 2016). In computer science, this is also known as “debugging.”

**Troubleshooting techniques**

1. **Identify**: The goal here is to clearly identify what is incorrect or what is the error.
2. **Isolate**: Locate the source of error by systematically testing the instructions, one by one.
3. **Reproduce**: Recreate the error to confirm that you have correctly identified the source of the error. This step also helps test solutions developed to remove the error.
4. **Fix**: Refine the instruction to correct the error and hence fix the issue.
Classroom activity

**Activity 1**: Have students choose two toys (one to be rescued, the other to be the hero), then have them practise this task, as follows.

1. Place the toy on a square on the edge of the grid, facing inwards.
2. Place the toy that is going to be rescued inside the grid.
3. The programmer writes down the program on a whiteboard.
4. The Tester then takes the whiteboard and a different coloured whiteboard pen. The Tester tells the bot each instruction in the program. The Tester puts a tick next to the code that is correct and underlines when the code is different to what the bot should be doing. If this happens, the Tester says, “Stop,” and the bot stops and goes back to the start. The Tester gives the whiteboard to the Programmer, who then debugs the code and gives the Tester a revised version.
5. Repeat step 4 until the program is free of bugs and works as intended.
6. Change roles, and move the bot toy and the toy that needs rescuing until everyone has had a turn.

* Adapted from CSUnplugged (n.d.).
**Activity 2**: Ask students to solve the Towers of Hanoi problem using computational thinking. Move the five coloured discs from Tower A to Tower B to recreate the disc pattern in Tower B. You can only move one disc at a time. No disc may be placed on top of a smaller disc on any of the towers. You can use Tower C for this activity, if needed.
1. While trying to solve a problem computationally, the first step is to
   a. Accurately identify and understand the actual problem that
      needs to be solved.
   b. Focus on how the problem needs to be solved rather than
      what needs to be solved.
   c. Tackle the complex problem as is, without breaking it into
      sub-problems.
   d. Guess what the solution might be.

2. Which of the following is true for patterns?
   a. Only shapes and colours can form patterns.
   b. Patterns consist of non-repeating sequences.
   c. More than one pattern can exist in solutions to problems.
   d. Irregular relationship between elements lead to very good
      patterns.
3. Which of the following is true for abstraction?
   a. It helps us gain additional information about a problem.
   b. It helps us focus on the important parts of a problem.
   c. It helps us find patterns.
   d. Knowledge of computers is essential to do abstraction.

4. Which of the following is true for algorithm writing?
   a. An algorithm consists of an ordered sequence of instructions.
   b. Algorithms prevent replication of a process.
   c. There can be only one algorithm for a problem.
   d. Knowledge of computers is essential to write an algorithm.

5. Which of the following is NOT a computational thinking practice?
   a. Programming
   b. Decomposing
   c. Abstraction
   d. Pattern identification

6. Which of the following needs to be kept in mind while designing an algorithm?
   a. The appropriate programming language needs to be used.
   b. It is correct the first time.
   c. It should not look like a recipe.
   d. More than one solution is possible.
7. Which of the following contains a pattern?
   a. I have two ears.
   b. All humans have ears.
   c. My friend wears earrings.
   d. My friend has two ears.

8. Which of the following will happen if we do not look for patterns?
   a. We will not be able to decompose the problem.
   b. Our solution might be incorrect and inefficient.
   c. Our solution will be inefficient.
   d. We will fail to solve the problem.

9. To be able to explain why a car skids, which of the following characteristics can be ignored?
   a. Whether it has good threaded wheels
   b. Whether it has doors
   c. Whether the surface is wet or icy
   d. Whether the car is speeding

10. Which of the following can be ignored while trying to understand how analogue clocks work?
    a. Clocks have a minute hand and an hour hand.
    b. The clock struck 2pm.
    c. The minute hand moves faster than the hour hand.
    d. Clocks have 1–12 numbers written on them.
11. You are preparing for a football match, and your captain starts briefing the team about past successful and unsuccessful strategies against the opposition. What is this an example of?

   a. Algorithm design
   b. Pattern recognition
   c. Decomposition
   d. Troubleshooting
Lesson 4

Modelling and Systems Thinking
Introduction

Models are simplified representations of reality that highlight important features that need to be attended to in order to understand, explain and predict reality. The use of models is very common in science (e.g., models of the water cycle, seismic activity, climate change, etc.). However, anything can be explained better using models. Models can range from simple physical scale models (e.g., a toy car, which highlights the external features of a real car while hiding the internal mechanics) to scaled diagrams (e.g., a map of a country, highlighting political boundaries) and even abstract diagrams (e.g., a free-body diagram in physics, showing force acting on an object). In this lesson, we focus on understanding models and how systems thinking helps to support computational thinking.

Learning outcomes

On completion of this self-study lesson, you should be able to:

• define the variables of a computational model
• design a computational model for a given problem
• refine a computational model
• explain how the model works
• develop a systems-thinking approach to solve problems computationally
What are computational models?

Computational models and simulations are dynamic representations of phenomena that can be simulated by a computer (Weintrop, 2016). Such models allow us to very quickly investigate questions and test multiple hypotheses that may otherwise be dangerous, expensive or impossible to do.

The following three steps outline a strategy to help students create and use models (Damelin et al., 2019).

1. Define the boundaries of the model: Think of what variables and components need to be represented in the model.
2. Design and construct the model structure: Highlight the relationship between the various variables and components.
3. Test, evaluate and revise the model behaviour: If the output of the model does not match the expected results, then refine the model.
We will now explore how to create and use computational models using the problem of determining how many people will catch a virus during an epidemic.

**Define the boundaries of the model**

Let’s now list down the variables that need to be represented in the model for solving this problem.

Susceptible people in the population: The number of people who are most likely to get infected due to age, prior health ailments, diet, living condition, etc.

Infectivity of the disease: The ease with which the disease spreads between people.
Design and construct the model structure

Now, let’s use the variables listed above and construct the model by highlighting relationships between them. One possible model for this problem of determining how many people will catch a virus during an epidemic is shown Figure “A simple model for determining the spread of an epidemic”.

As presented in the example above, a typical model shows the variables and their interconnections and influence on each other (represented by the plus and minus signs showing positive and negative effects on the connected variable). The above model shows that increased “number of vaccinations” using “effective vaccines” add together to determine the “effectiveness of a vaccination program” (symbolised here by the green check mark).
This vaccination program then reduces the number of new infections (indicated by the minus sign on the arrow connecting the “vaccination programme” and “number of new infections”). However, new infections are also increased by the cumulative effect of “number of susceptible people in the population,” “number of people who come in contact with each other” and the “infectivity of the disease.” These variables, together with the “vaccination program,” help determine the “number of infected people.”

A simple model for determining the spread of an epidemic
(Source: HYPERLINK "Epidemic – static model", developed using SageModeler [2018], created by Concord Consortium.)

However, this model does not take into consideration feedback — i.e., the effect of “number of infected people” on “number of new infections” by various means, such as social events where infected and non-infected people intermingle with each other. The occurrence of such events will likely lead to an increase in the “number of new infections” and hence an increase in the “number of infected people.” Thus, this initial model needs to be refined further to represent reality and accurately predict the answer to the problem of determining how many people will catch a virus during an epidemic.
Test, evaluate and revise model behaviour

After the design stage, one must evaluate and test a model to see whether it accurately predicts and explains the real-world scenario. For instance, in Figure “A simple model for determining the spread of an epidemic”, if after entering the values for various variables in the model, one finds that the number of infected people keeps on increasing, but in reality, the number of infected individuals gradually decreases over time, then the model is not accurate. Here, we are testing the initial model by entering values and evaluating the results of the model to determine its accuracy.

As discussed in the previous section, our initial model of the epidemic (Figure “A simple model for determining the spread of an epidemic”) will not give us accurate results, since we have not considered feedback (the effect of infected people on the number of new infections). Thus, testing and evaluation reveals that our model in Figure “A simple model for determining the spread of an epidemic” is missing some key variables that affect the model’s output. A revised version might look like Figure “Refined model for determining the spread of an epidemic”.

Refined model for determining the spread of an epidemic

(Source: "Epidemic – dynamic" model, developed using SageModeler [2018], created by Concord Consortium.)
We test this revised model again to check for accuracy. We can plot graphs to determine the model’s behaviour over time. Graphs help us understand patterns and make abstract conclusions from the model. For example, in Figure “infected individuals” and “days”, “Refined model for determining the spread of an epidemic”, we see that the number of “infected individuals” increases rapidly initially but then starts to gradually reduce after approximately 20 days.

![Graph plotted for “infected individuals” and “days,” using revised model in Figure “Refined model for determining the spread of an epidemic”.](Source: "Epidemic – dynamic model", developed using SageModeler [2018] created by Concord Consortium.)
To explain this outcome, we can plot further variables on the graph to understand their influence on each other. For instance, the gradual reduction in number of infected individuals could be due to an effective vaccination programme and/or also due to reduced chances of contact with infectious people (Figure “infected individuals,” “chances of contact being infectious” and “days,”).

Graph plotted for “infected individuals,” “chances of contact being infectious” and “days,” using revision of model in Figure “Refined model for determining the spread of an epidemic”.

(Source: "Epidemic – dynamic” model, developed using SageModeler [2018] created by Concord Consortium.)

Using such models, one can thus explain and solve problems pertaining to determining the spread of an epidemic. We can also ask exploratory questions such as “What would happen if the number of people wearing masks increased?” or “What if the infectivity of the disease increased?” Such questions can lead to further iterative refinement of the model, thereby ensuring greater accuracy.

The two graphs above have been created using SageModeler (2018). One can also use Excel or any other computational tool.
We have not implemented the model yet; instead, we are still revising the model to increase its accuracy. Implementation should happen only after a model has been designed and constructed accurately through careful evaluation, and the corresponding algorithm has been written for implementation. The implementation stage may require a knowledge of computer programming.

Classroom activity

**Activity 1:** Ask students to identify further variables that could be part of the above epidemic model. Students should connect these additional variables to existing ones and then test and evaluate their revised model.

**Activity 2:** Ask students to create a model representing Little Red Riding Hood’s problem and the related navigation system. What variables should be part of this model? How are these interconnected?
Computational Thinking for Teachers

In the following video, we will revisit what we have learned about modelling in the above sections. We will focus on what is meant by computational modelling and systems thinking, and how to create and use models.

Watch Video: https://www.youtube.com/watch?v=cncnYOlbXYs

Video attribution: "Modelling and Systems Thinking" by Commonwealth of Learning is available under CC BY-SA license.
In the preceding sections, we have been thinking of solving the problem of determining the spread of an epidemic. The problem was represented in the form of a system that had various components or variables, and we were trying to understand this system’s behaviour by recognising the interconnections between the components or variables. Thus, we have been engaging in systems thinking. Systems thinking is a holistic approach to problem solving that emphasises the need to focus on a system’s components and how these interrelate and work together to give rise to the whole system’s functionality. This emphasis on interconnectedness leads to problem solving from the perspective of causality — how one thing dynamically influences another thing in a system. The computational tools we have discussed up to this point make systems thinking accessible and easily understandable for learners.

**Why and how: a systems-thinking perspective for solving problems computationally**

When we are solving problems using computational thinking, we borrow from systems thinking practices such as defining the boundary or scope of a problem or system and recognising connections between components or variables of the problem or system. These systems thinking practices are coupled with specific computational thinking practices.
For instance, characterising the problem involves decomposing the problem, which is a computational thinking practice. Similarly, while designing an abstract model, students bring together the need to model the complete system from a systems-thinking perspective and test and debug this model from a computational thinking perspective.

“The ability to think systemically is an important habit of mind that supports not only the scientific background of the developing STEM workforce, but also future scientifically literate citizens. In a global society where future large-scale, scientifically based decisions will need to be made, it is important for the general populous to develop a systems thinking orientation toward the world” (Duschl and Bismack 2013, p. 120).
Figure “Adapted framework of system modelling that combines aspects of systems thinking and computational thinking” summarises the connections between computational thinking and systems thinking and how the latter helps with solving problems computationally.

This framework helps us characterise every problem as a system, thus enabling holistic problem solving. To characterise a problem or phenomenon, we first decompose the problem into sub-problems that can be solved computationally. Next, we identify the variables involved and their causal interrelationship and determine the scope of the problem or boundary of the system — i.e., which variables or components do not matter for solving the problem.
We observe patterns and create abstractions relevant for designing, constructing and revising the model. We iteratively refine these patterns and abstractions, and hence the model, by testing, evaluating and revising the model. Here, we use algorithms by going through the model sequentially, beginning with the input variables, following a set of steps that lead to the outcome from the model. We troubleshoot the algorithm and the model to detect any errors or inaccuracies. The final refined model is then used to explain and predict the behaviour of the real phenomenon or design a solution to the problem.
Classroom activity

Let’s try to bring it all together now. Ask students to work in small groups to map the Red Riding Hood problem as a system modelling cycle, as in Figure “Adapted framework of system modelling that combines aspects of systems thinking and computational thinking”. Conduct a class discussion wherein the small groups can collate their ideas, sketches and write-ups. Students can use their work up to this point and reorganise/refine it while connecting ideas together. They can either create storyboards reflecting the problem-solving process and intermediate solutions, or write a sequential list of steps, or even mimic the flowchart in Figure “Adapted framework of system modelling that combines aspects of systems thinking and computational thinking” and place their work next to the boxes. You may simplify this activity as required to meet your grade-specific needs and challenges.
1. Which of the following is true for computational models?
   a. They are dynamic, simplified representations of reality, simulated using computers.
   b. They show parts of a computer.
   c. They are used for explaining how computers work.
   d. They are imaginary scenarios that do not represent reality.

2. What is a model?
   a. A model is a computer program.
   b. A model is a spreadsheet.
   c. A model is a representation of a problem.
   d. A model is an imaginary concept.
3. Which one is a computational modelling strategy?
   a. Wait and watch.
   b. Define the boundaries.
   c. Guess and check.
   d. Seek help from an expert.

4. Which of these is important while creating a computational model?
   a. List only a few variables in the system.
   b. Show only a few connections between the variables.
   c. Show causal relationships between all the variables.
   d. Figure out which programming language to use.

5. Which one of the following statements is true?
   a. It is important to test and evaluate a model.
   b. There is always only one right model.
   c. Iterative refinement of a model should not be done.
   d. A lot of random trial and error is involved in the creation of a model.

6. A computational model
   a. Is always accurate.
   b. Is a representation of reality.
   c. Helps in guessing the solution.
   d. Is the same as a computer program.
7. A graph is useful because
   a. It helps in interpreting the output of a model.
   b. It is a visual way of showing the variables in a model.
   c. It lets one be creative.
   d. It has two axes — x and y.

8. Systems thinking is important in computational thinking because
   a. It leads to a holistic problem-solving approach.
   b. It helps keep the system’s boundaries vague.
   c. It helps with guess and check.
   d. It helps us think about the computer system needed for solving the problem.

9. As a teacher, which one would you do if you were practicing systems thinking?
   a. Always focus on a few students in the class.
   b. Consider all the students’ prior knowledge and resources available while designing a lesson plan.
   c. Ignore the availability of resources for each student.
   d. Think of the classroom as a disconnected space that is not affected by the school and the district’s culture.

10. As a teacher, which one would you do if you were practicing computational thinking?
    a. Make students solve complex problems without breaking them down into smaller manageable chunks.
    b. Identify the high performers and just teach them.
    c. Identify important factors that affect students’ performance, interconnect these and come up with a list of steps to help students make progress in class.
    d. Ignore students’ feedback.
References

https://www.bbc.co.uk/bitesize/topics/z7tp34j

https://www.computerhope.com/jargon/d/datamani.htm

https://www.cse.iitb.ac.in/~sri/ssrvm/

https://sagemodeler.concord.org

https://community.computingatschool.org.uk/resources/2324/single

CS4FN. (n.d.). *Computer science for fun: Interactive applets for computational thinking*. 
http://cs4fn.org/algorithms/swappuzzle/

https://csunplugged.org/en/topics/kidbots/unit-plan/rescue-mission/


Digital Promise. (n.d.). *Data practices*. 
https://digitalpromise.org/initiative/computational-thinking/computational-thinking-for-next-generation-science/data-practices/


References


IgniteMyFuture. (n.d.). *K12 computational thinking resources.* [https://www.ignitemyfutureinschool.org/resources/k12-computational-thinking-resources](https://www.ignitemyfutureinschool.org/resources/k12-computational-thinking-resources)


Vanderbilt University School of Engineering. (n.d.). *CTSim.* Open Ended Learning Environment. [https://wp0.vanderbilt.edu/oele/ctsim/](https://wp0.vanderbilt.edu/oele/ctsim/)


Image Attributions

Figure “Abstraction of a coffee machine”: “India physical map, India administrative map” by Wikimedia Commons is licensed under CC BY-SA 3.0.

Figure “Algorithm designed to check whether a lamp is broken”: “Lamp flowchart” by Wikimedia Commons is licensed under CC BY-SA 3.0.

Figure “Warli art”: “A Warli painting” by Jivya Soma Mashe is licensed under CC BY 2.0.

Figure “Painting by Australian aboriginals”: “Aboriginal Art Aboriginal Painting” by Pixabay, licensed under Pixabay License – free to use for commercial and non-commercial purposes.

Figure “A simple model for determining the spread of an epidemic”: Screenshot of “Epidemic – static” model, open-source, free-to-use SageModeler software created by Concord Consortium.

Figures “Refined model for determining the spread of an epidemic”, “infected individuals” and “days”, “infected individuals,” “chances of contact being infectious” and “days,“: Screenshot of “Epidemic – dynamic” model, open-source, free-to-use SageModeler software created by Concord Consortium.

Figure “Adapted framework of system modelling that combines aspects of systems thinking and computational thinking”: Adapted from “Engaging in Computational Thinking Through System Modeling” (2019), by Dan Damelin, Lynn Stephens, and Namsoo Shin from Concord Consortium and Michigan State University.
Lesson 1

(1) d , (2) b , (3) c ,
(4) a - Abstraction is the process of filtering out irrelevant characteristics and unnecessary details.
(5) b
(6) b - We must construct a sequence of instructions or set of rules for designing the experiment so that an algorithm can be used.
(7) c , (8) b ,
(9) a - Bubble sort is the simplest sorting algorithm. If adjacent elements are in the wrong order, it works by repeatedly swapping them.
(10) a

Lesson 2

(1) b , (2) d , (3) b , (4) a , (5) d ,
(6) a - Rearranging is one method of data manipulation, usually requiring a repositioning of data elements so that the items of information we are seeking sit closer to each other.
(7) c ,
(8) c - The process of cleaning, modifying and analysing raw data to extract substantive, useful information is called data analysis.
(9) a , (10) b
Lesson 3

(1) a - It is essential to accurately understand the problem, since skipping this step or rushing to the solution may result in inefficient and ineffective solutions.

(2) c - Multiple patterns may emerge while solving a problem, since the same solution may have more than one pattern. Moreover, a problem can have multiple solutions.

(3) b - Abstraction removes irrelevant or less important information and prevents distraction.

(4) a - An algorithm comprises steps required to solve a problem or complete a task.

(5) a - Knowing how to write a program is not essential for practicing computational thinking. It is more important to know how to decompose a problem, abstract information and identify patterns.

(6) d - There can be multiple ways of solving a problem.

(7) b - The presence of ears in all humans is a repeating pattern.

(8) b - Patterns help us improve accuracy and efficiency when generating solutions.

(9) b - The presence of doors is irrelevant and not a factor causing a car to skid.

(10) b - The clock striking 2pm is irrelevant for understanding how a clock works, so this information can be ignored.

(11) b - By reviewing previous games and strategies, the captain of the team identifies actions and decisions that may be repetitive and hence form a pattern.
Lesson 4

(1) a - Models are simplified representations of abstracted information and can change dynamically according to changes in the input parameters.

(2) c - A model represents simplified information pertinent to a problem.

(3) b - It is important to define the boundaries of a model by considering what variables and components need to be represented in the model.

(4) c - It is important to show how one thing dynamically influences another thing in a system.

(5) a - After designing a model, one must evaluate and test the model to see whether it accurately predicts and explains the real-world scenario being examined.

(6) b - A model is a simplified representation of a real phenomenon.

(7) a - Graphs help us understand patterns and make abstract conclusions from a model.

(8) a - One must keep the entire system in mind while solving a problem computationally.

(9) b - Students’ prior knowledge is an important variable affecting how a lesson plan needs to be designed to support students’ needs.

(10) c - It is important to think about the entire system, identify the variables that matter and then determine how these affect one another.