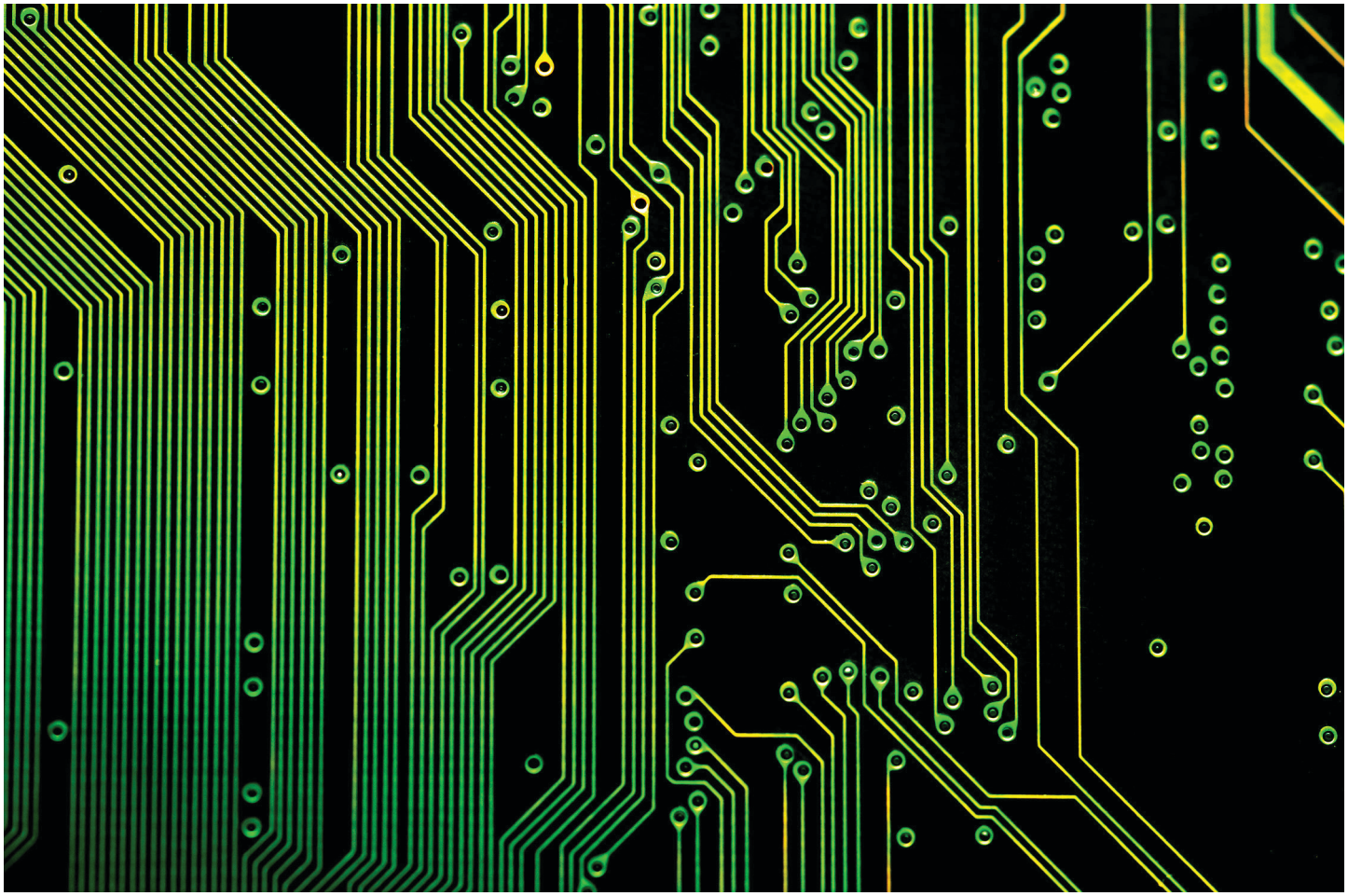
OpenLearn

The Open University

Introduction to computational thinking



About this free course

This free course is an adapted extract from the Open University course M269 *Algorithms, data structures and computability* http://www3.open.ac.uk/study/undergraduate/course/m269.htm.

This version of the content may include video, images and interactive content that may not be optimised for your device.

You can experience this free course as it was originally designed on OpenLearn, the home of free learning from The Open University: www.open.edu/openlearn/science-maths-technology/computing-and-ict/introduction-computational-thinking/content-section-0.

There you'll also be able to track your progress via your activity record, which you can use to demonstrate your learning.

Intellectual property

Contents

Introduction

> One can major [i.e. graduate] in computer science and do anything. One can major in English or mathematics and go on to a multitude of different careers. Ditto computer science. One can major in computer science and go on to a career in medicine, law, business, politics, any type of science or engineering, and even the arts.
>
> Jeannette M. Wing, Professor of Computer Science at Carnegie Mellon University (United States) and Head of Microsoft Research International

Sounds great that 'One can major [i.e. graduate] in computer science and do anything', doesn't it? Then again, isn't this miles away from the view of computing as a training ground for programmers and system builders? The good news is that one doesn't necessarily need to exclude the other! The grand vision behind this quote is that learning to program, build large systems, and so on, allows you to develop something which is much more valuable than any of these on their own, namely the ability to *think like a computer scientist*. Over the past decade or so, Jeannette Wing has been popularising this view under the banner of *computational thinking*.

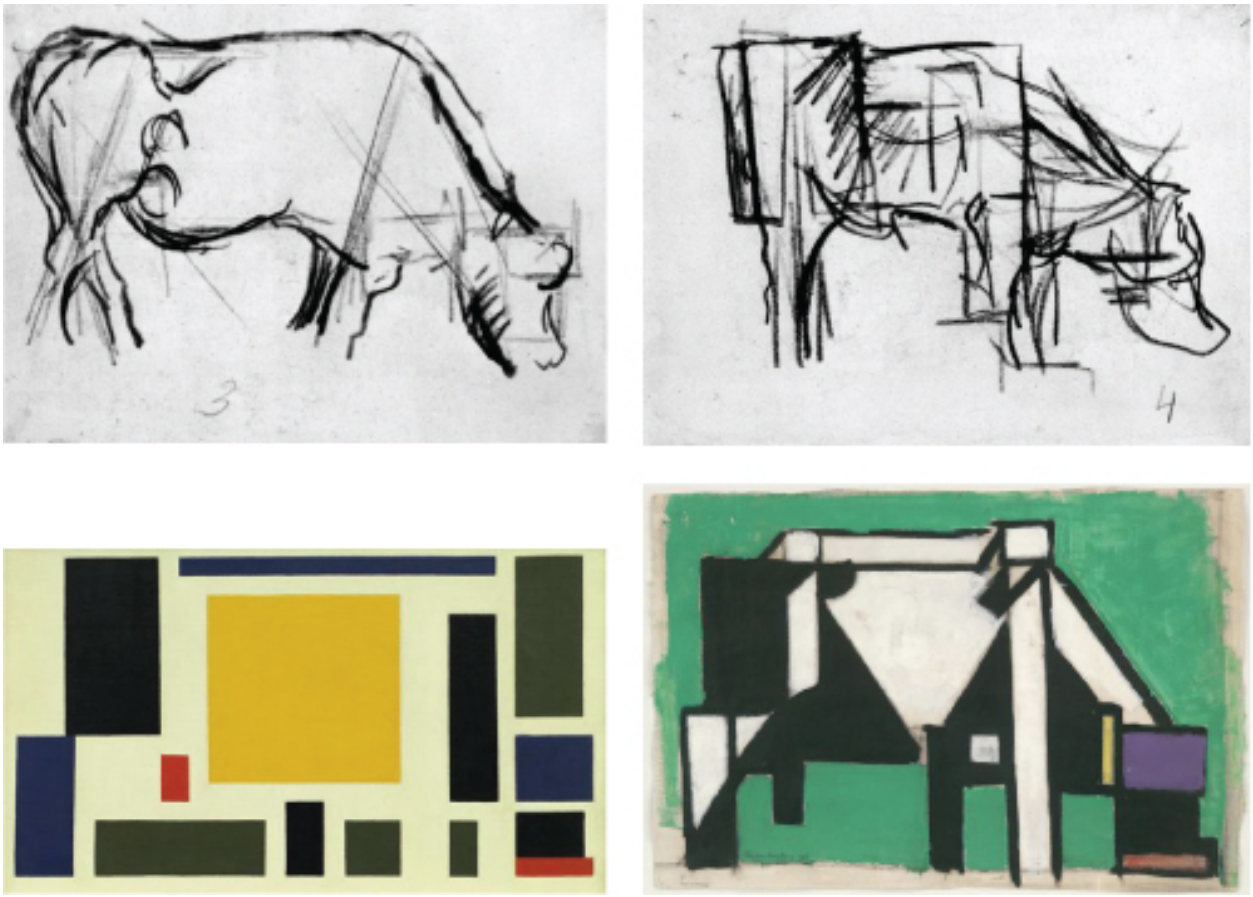


Figure 1 Abstraction, a recurring theme in this course, illustrated by four representations of a cow by Theo van Doesburg, 1917, 1918

Much of the material in this course is organised around video clips from a presentation that Wing gave in 2009 entitled 'Computational Thinking and Thinking About Computing'

(Wing, 2009). The presentation builds on Wing's influential 2006 'Computational Thinking' paper in which she set out to 'spread the joy, awe, and power of computer science, aiming to make computational thinking commonplace' (Wing, 2006, p. 35).

In this course you will learn more about what computational thinking is and why it is such a desirable skill – arguably *the* skill for the twenty-first century.

This OpenLearn course is an adapted extract from the Open University course M269 *Algorithms, data structures and computability*.

Learning Outcomes

After studying this course, you should be able to:

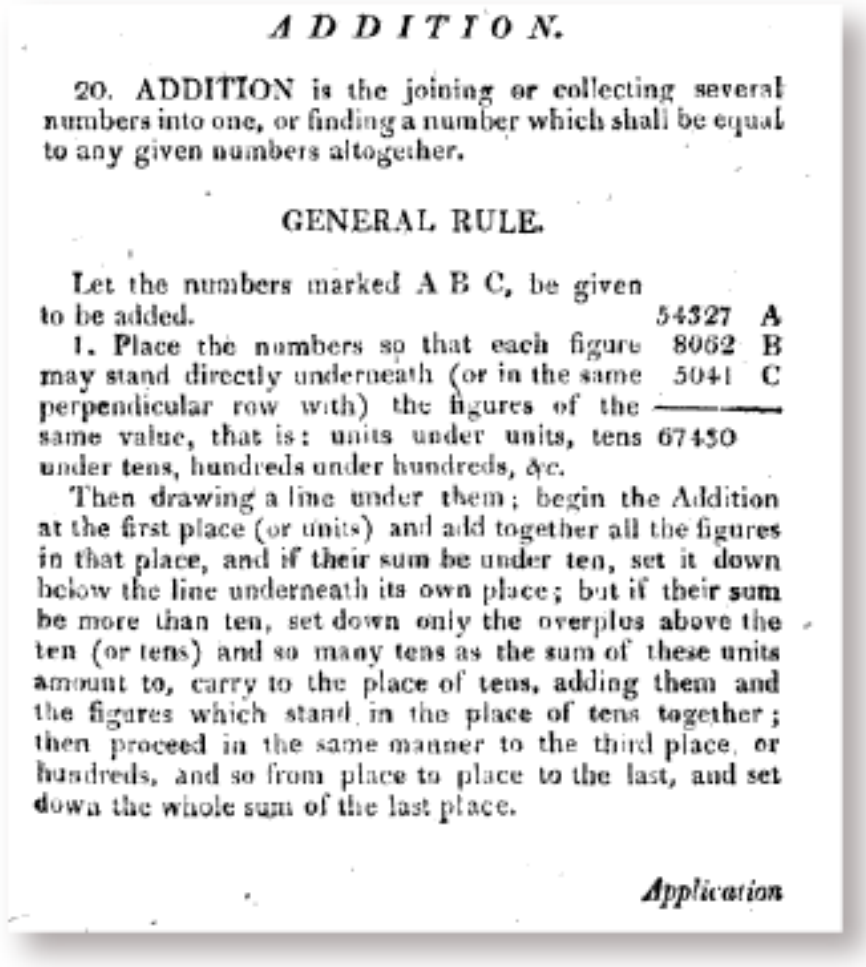
- describe the skills that are involved in computational thinking
- define and use the concepts of abstraction as modelling and abstraction as encapsulation
- understand the distinctive nature of computational thinking, when compared with engineering and mathematical thinking
- be aware of a range of applications of computational thinking in different disciplines.

1 Computational thinking and automation

To state more precisely what computational thinking is, we need to consider the ideas of an algorithm and a computational problem. We will think about these ideas in the next activity.

Activity 1 Algorithms and computational problems

Look at Figure 2. This shows page 18 of John Gough's *Practical Arithmetick in Four books*, which is a tutorial maths book first published in 1767. The text under the heading 'General Rule' gives a description of a process for adding together three non-negative integers, each written as a finite sequence of digits. (The fact that three numbers are added is rather arbitrary: the process would work just as well for adding together two numbers or indeed any number of numbers.)



ADDITION.

20. ADDITION is the joining or collecting several numbers into one, or finding a number which shall be equal to any given numbers altogether.

GENERAL RULE.

Let the numbers marked A B C, be given to be added.

1. Place the numbers so that each figure may stand directly underneath (or in the same perpendicular row with) the figures of the same value, that is: units under units, tens under tens, hundreds under hundreds, &c.

54327	A
8062	B
5041	C
67430	

Then drawing a line under them; begin the Addition at the first place (or units) and add together all the figures in that place, and if their sum be under ten, set it down below the line underneath its own place; but if their sum be more than ten, set down only the overplus above the ten (or tens) and so many tens as the sum of these units amount to, carry to the place of tens, adding them and the figures which stand in the place of tens together; then proceed in the same manner to the third place, or hundreds, and so from place to place to the last, and set down the whole sum of the last place.

Application

Figure 2 Instructions for column addition (Gough, 1767)

Explain whether this textual description is an algorithm according to the following description of an algorithm.

An algorithm is a solution to a problem which has three properties:

- an algorithm consists of a step-by-step list of instructions
- an algorithm is a finite process (this means it is guaranteed to finish at some point)
- the algorithm should solve any instance of the problem that might arise.

Discussion

First, we need to consider whether the text describes a step-by-step list of instructions. This is not always obvious. At some points it does and at some points it doesn't.

At many points the text does provide instructions to be carried out sequentially, such as:

- 'place the numbers so that each figure may stand directly underneath (…) the figures of the same value'
- 'begin the addition at the first place'.

Although other parts of the instructions are not explicitly presented as a list of instructions, it is easy to see how the textual passage could be mapped onto such a list. In that list, certain instructions would be repeated for each of the columns involved ('then proceed in the same manner … from place to place to the last').

So, let's assume that the text does have the first property. Does it also have the other two properties?

According to the second property, the process should be finite. This is indeed the case. The numbers being added will each have a finite number of digits, so only a finite number of columns will be involved, and the process only requires a small number of steps at each column.

Finally, the process does apply to any instance of the problem, where the 'instances of the problem' are all combinations of three integers, each represented by a finite sequence of digits in the range 0–9. The text leaves it implicit what instances the algorithm is intended to apply to.

The precise specification of the 'instance of the problem' can be thought of as part of the task of formulating a problem as a computational problem. By a computational problem, we mean "a problem that is stated sufficiently precisely such that one can attempt to write an algorithm to solve it."

Figure 3 (a) Human computers working on test flight calculations at the predecessor of NASA in 1949 (b) two NASA employees in 1990 programming a computer so it can automatically calibrate aircraft equipment

Having met the ideas of algorithms and computational problems, let us state what computational thinking is not: computational thinking is *not* about us humans following an algorithm when carrying out the task of adding numbers on paper or in our head. It is not about thinking *like* a computer - rather, computational thinking is first and foremost thinking *about* computation. In particular, computational thinking consists of the skills to:

- formulate a problem as a *computational problem*
- construct a good computational solution (i.e. an algorithm) for the problem, or explain why there is no such solution.

A computational thinker can take a problem and state it sufficiently precisely for it to be potentially solvable by an algorithm. Once the problem has been stated in the right way, a computational thinker tries to construct an algorithm that solves it. A computational thinker won't, however, be satisfied with just any solution: the solution has to be a 'good' one. We will discuss two specific properties of a good solution later on in this course: efficiency and correctness. Finally, computational thinking goes beyond finding solutions: if no good solution exists, one should be able to explain why this is so. This topic, however, takes us beyond the scope of this course into computability and computational complexity theory.

Most of the activities in this course involve watching extracts from Wing's presentation 'Computational Thinking and Thinking About Computing'. Remember to read through the activity before watching the video.

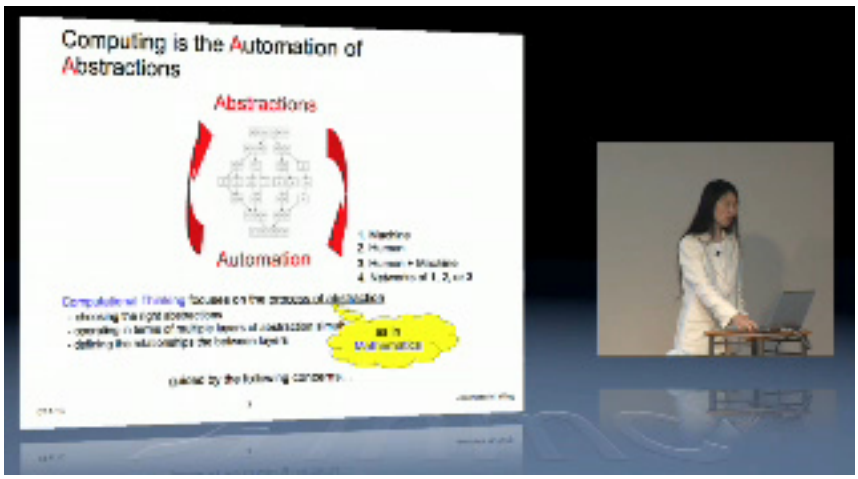
1.1 Automation

Activity 2 Automation, abstraction and modelling

5 minutes

After you've watched the following extract of Wing's talk, complete this activity.

Video content is not available in this format.
Automation, abstraction and modelling



The aim of this activity is to get you to think about what Wing is saying in the following, rather long but important, sentence: '… this feedback loop that one has when you're abstracting from some physical-world phenomenon, creating a mathematical model of this physical-world phenomenon, and then analysing the abstraction, doing sorts of manipulations of those abstractions, and in fact automating the abstraction, that then tells us more about the physical-world phenomenon that we're actually modelling.'

Place the following labels in the correct positions of the diagram below. (**Note**: the diagram will not work with Internet Explorer 8, if this is your usual browser you will need to use an alternate browser to view the diagram.)

Interactive content is not available in this format.

Discussion

It is important to have a good grasp of this diagram. It provides a high-level overview of the key ingredients of computational thinking and how they are related to each other. In the remainder of this course, we will flesh out some of the details that lie behind this diagram.

2 Computational thinking and abstraction

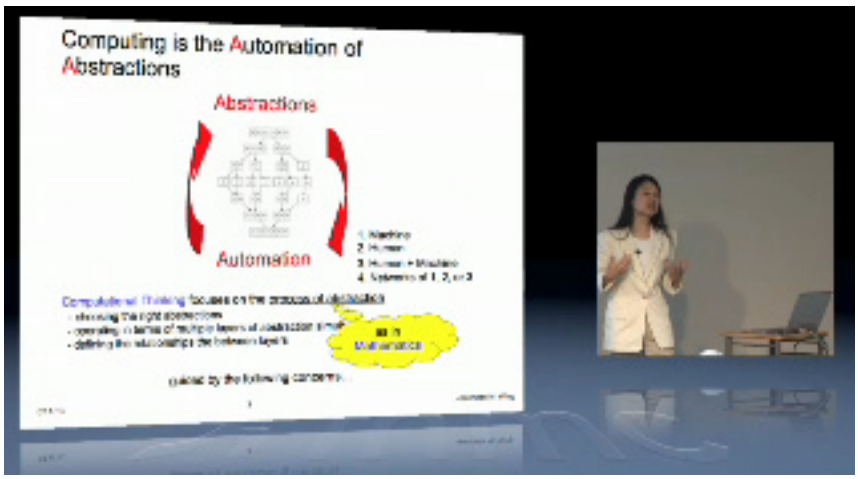
In the next activity you will see Wing explain how computational thinking involves the process of abstraction.

Activity 3 Abstraction

8 minutes

Watch the video and then complete this activity.

Video content is not available in this format.
Abstraction



Wing talks about abstractions as a means of ignoring detail that is not of interest. Consider the following description of an algorithm for searching a paper and ink dictionary.

Given:

(i) a dictionary

(ii) a headword *W* which appears in the dictionary

to find the definition of *W*:

Begin the search with an entry that appears (roughly) in the middle of the dictionary. Then there are three possibilities:

1 The current entry's headword is *W*. In this case, the definition of *W* is the definition of the current entry.

2 *W* appears alphabetically before the current entry's headword. In this case, the entry with headword *W* must appear in the first half of the dictionary. Look for the word *W*, following the same approach, in the first half of the dictionary.

3 *W* appears alphabetically after the current entry's headword. In this case, the entry with headword *W* must appear in the second half of the dictionary. Look for the word *W*, following the same approach, in the second half of the dictionary.

This is an informal description of an algorithm, but the steps are sufficiently precise that a human should have no problem following them. This algorithm depends crucially on the fact that the words in a dictionary are alphabetically ordered. It also, however, ignores some of the details of paper and ink dictionaries. Can you list of few of these?

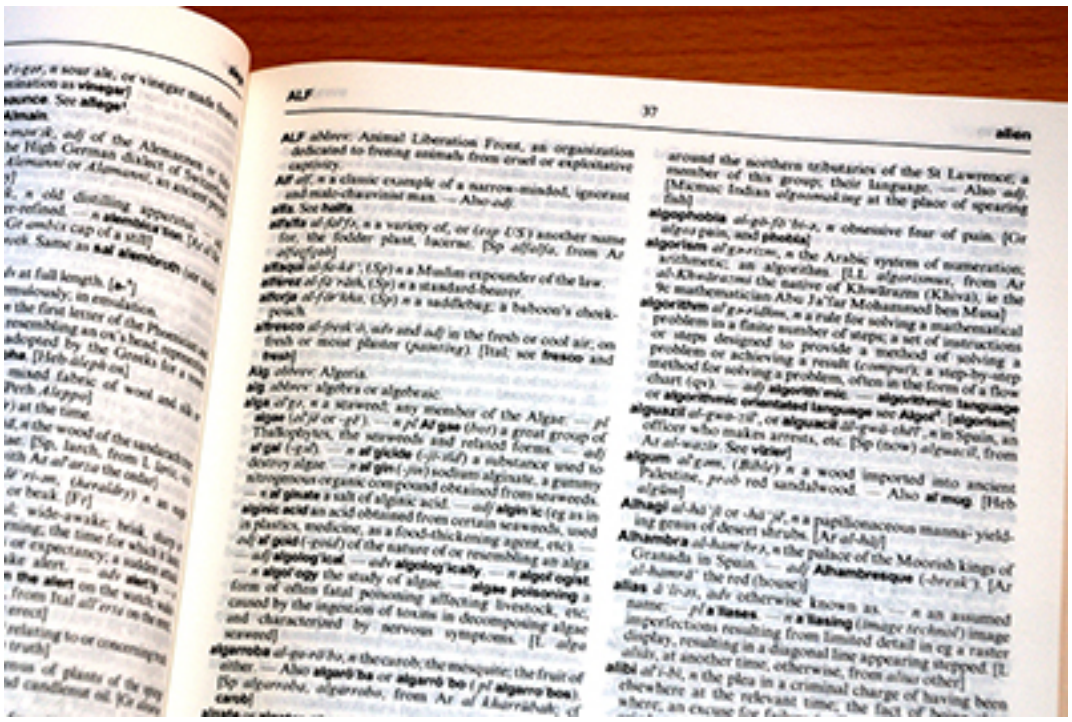


Figure 4 A dictionary

Discussion

Details of a paper and ink dictionary that come to mind include: the size and thickness of its pages, the colour of its cover and the size of the printed font. None of these played a role in the algorithm that is described above. Perhaps the most salient abstraction relates to the fact that paper and ink dictionaries are made of pages. They are not simply a list of words and definitions. Rather, the words are distributed across the pages of the dictionary. The algorithm doesn't take into account the distribution of the words in a dictionary across pages: for example it would work equally well with a dictionary printed on a single, enormously long paper scroll.

Wing's discussion of **abstraction** draws on a wide variety of concepts. You have come across phrases such as 'ignoring' and 'hiding information', 'levels' and 'layers', 'models' and 'implementation'. It has been left somewhat in the air how these all hang together. So next, we will make the relations between them explicit. To do so, we will distinguish between two kinds of abstraction:

- abstraction as modelling
- abstraction as encapsulation.

Although Wing deals in great detail with the idea of abstraction as modelling, she only hints at the concept of abstraction as encapsulation. Both are, however, at play in computing. Distinguishing between them will give you a better understanding of the remainder of Wing's story.

2.1 Models

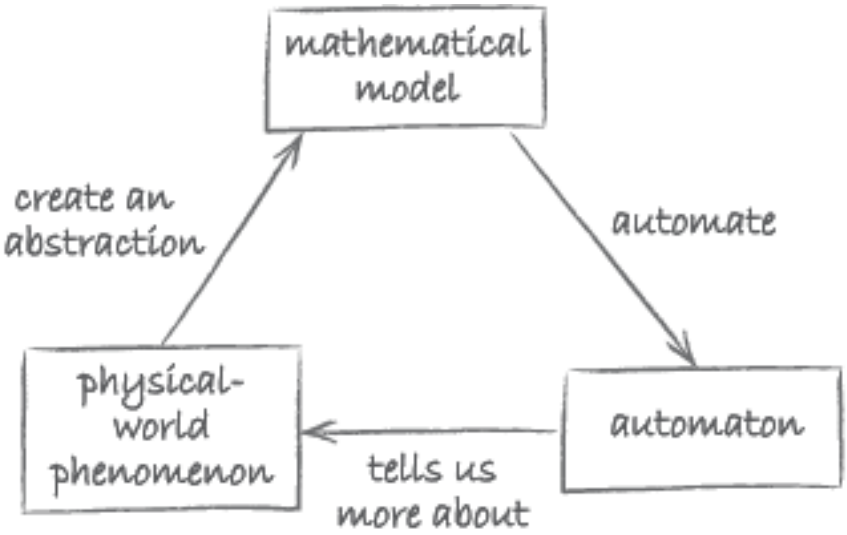


Figure 5 Abstraction and automation

Let us think more generally about the abstraction depicted in Figure 5. For this purpose, we will be using the diagram in Figure 6. Instead of Wing's somewhat unwieldy expressions 'mathematical model' and 'physical-world phenomenon', we will use the shorthands 'model' and 'part of reality'.

The diagram in Figure 6 shows that abstraction as modelling can be understood in terms of the relationship between a part of reality and a **model** which represents the details of interest of this reality. For this reason, models are sometimes also referred to as representations.

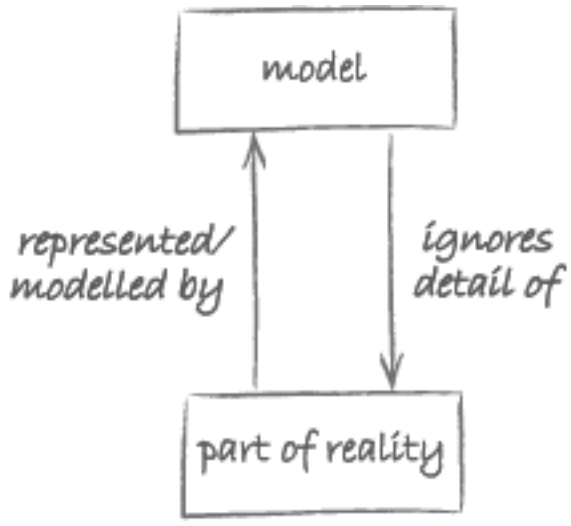


Figure 6 Diagrammatic representation of abstraction as modelling

For the purpose of the current example, the part of reality that we're interested in is the paper and ink dictionary of Activity 3. The algorithms in Section 1 for looking up words work with a model of a dictionary as an alphabetically sorted list of headwords, each paired with its definition. The model corresponds to how the data that the algorithm works with are structured. This model *captures the essentials* (i.e. that the dictionary consists of

an alphabetically sorted list of words and their definitions) and *ignores certain details* (for example, that a physical dictionary is divided into pages of a certain size and has a cover to protect it). In other words, the irrelevant details are *not* reflected by the structure of the data that the algorithm works with.

Activity 4 Representations

Representations, including paintings and drawings, are models. This is brilliantly illustrated by René Magritte's famous painting of a pipe with the words 'Ceci n'est pas une pipe' ('This is not a pipe') underneath (Figure 7). Explain how this painting brings home the point that modelling involves *two* levels: the abstraction and the reality which the abstraction models.

Figure 7 René Magritte's painting *The Treachery of Images* (1928–9)

Discussion

When looking at this painting, my initial thought was: 'Look, a pipe – why does it say in French that this is not a pipe?' Then it dawned on me that of course this is indeed not a pipe: it's merely a painting of a pipe. With this painting, Magritte reminded me that representations usually involve two levels: in this case the painting of a pipe and a real pipe on which the painting is based. The former is an abstraction which ignores some of the properties of actual pipes (for example, that they have a smell, usually of tobacco). Interestingly, the title of this painting is *La trahison des images* (*The Treachery of Images*).

The part of reality for which one creates a model or representation doesn't need to be a static object such as a pipe or dictionary. It can also be a process or system which changes over time. In that case, the model will not only have to capture any enduring properties of interest, but also any changes in time that are of interest. For example, in astronomy there is a rich tradition of building such models of the solar system (see Figure 8). The purpose of such a model, known as an orrery, is to represent the position of the planets relative to each other over time.

The invention of the orrery is usually attributed to George Graham (1673–1751). Graham worked on this with the support of the 4th Earl of Orrery, which explains the name. The tiny crank on the side of the central cylinder in Figure 8 allows the model to be animated. Turning the crank causes the (representations of the) planets to follow a path around the (representation of the) Sun at the centre of the model. Usually such an animated model is referred to as a **simulation**.

Figure 8 An orrery (a mechanical model of the solar system)

As with any model, an orrery ignores certain details of the part of reality that it models. For example, compared with the scale of the distances between the planets, the planets are too large. The relationship between the solar system and an orrery is as depicted in Figure 9.

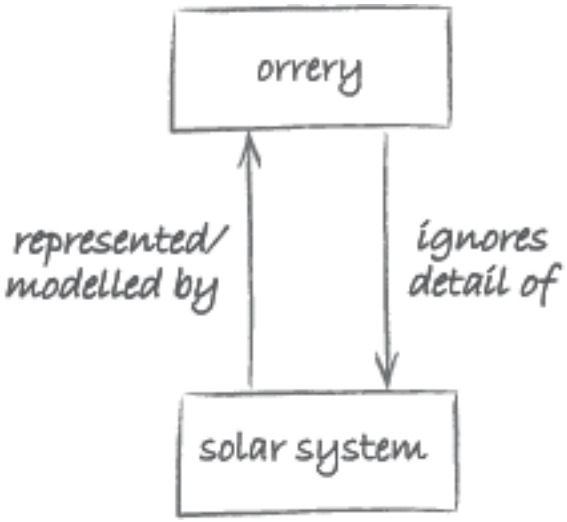


Figure 9 Relationship between an orrery and the solar system

2.2 Encapsulation

The model of the solar system in Figure 10 also illustrates the second type of abstraction: abstraction as encapsulation. The brass cylinder in the middle of the device encloses a clockwork of cogwheels similar to the one in Figure 10.

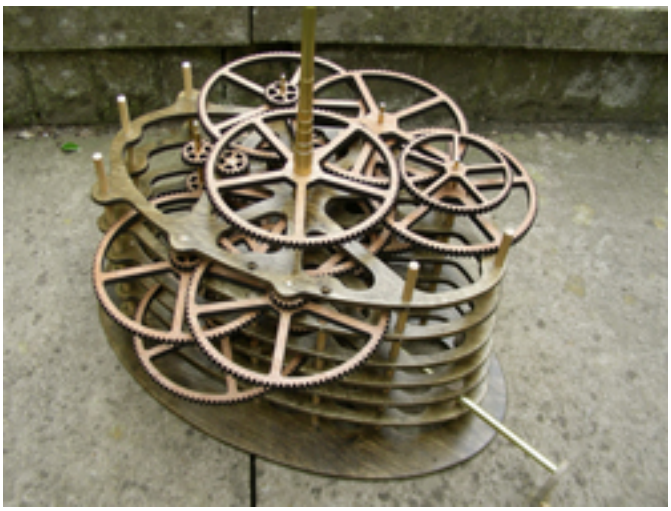


Figure 10 The inside of an orrery

This mechanism causes the small spheres representing the planets to rotate around the centre at a velocity that is proportional to the speed of the actual planets. The mechanism is hidden from view for the casual user, who is only interested in the movement of the planets. Figure 11 illustrates this other view of the orrery in terms of two layers.

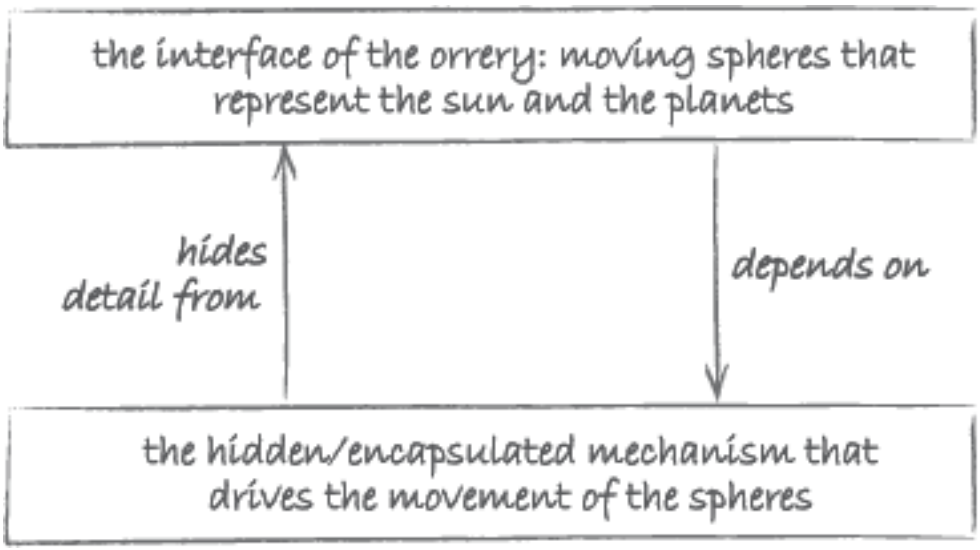


Figure 11 The orrery as two layers

Generally, abstraction as encapsulation involves the two layers shown in Figure 12.

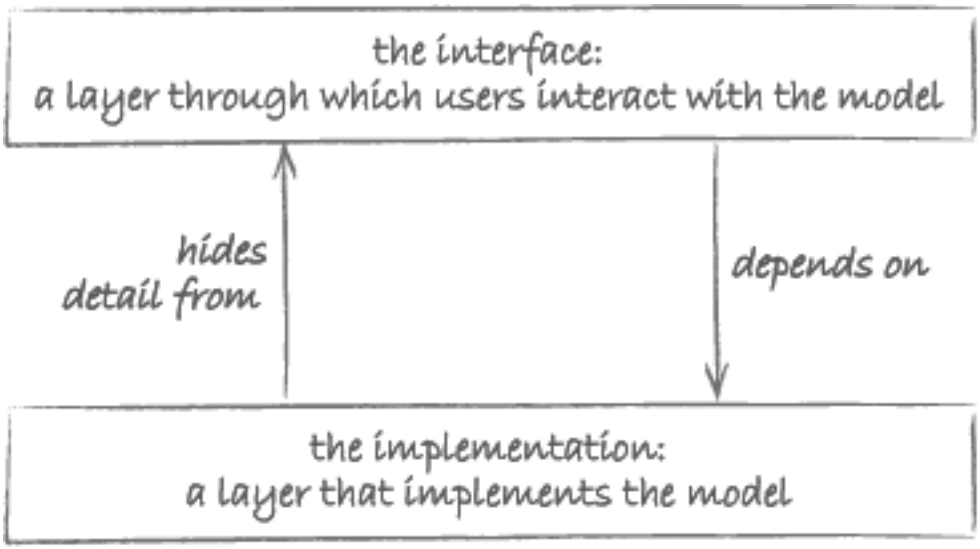


Figure 12 The two layers involved in abstraction as encapsulation

The layer through which the user interacts with the model is called the **interface**. It hides the detailed workings of the model from the user. The interface sits between the user and the layer at which the model is implemented. The latter is responsible for making the model do what it is supposed to do. This is where the automation of the model takes place.

2.3 Encapsulation in computing

Abstraction as encapsulation is common everywhere in computing. Sometimes it is referred to as 'abstraction as information hiding' (instead of encapsulation).

Let us consider an example. Our example will make use of the Python programming language. This language is used throughout M269 *Algorithms, data structures and computability* on which this OpenLearn course is based. It was chosen because it is eminently suitable for learning the basic concepts of computer science and programming. It allows you to write concise, easily understood programs. You should be able to follow most of the following discussion if you have had some experience with an imperative programming language.

The incredibly short Python program below will display the date and current time (see Figure 13). It depends on a built-in Python module which is imported in the first line.

```
import time
print(time.ctime())
```

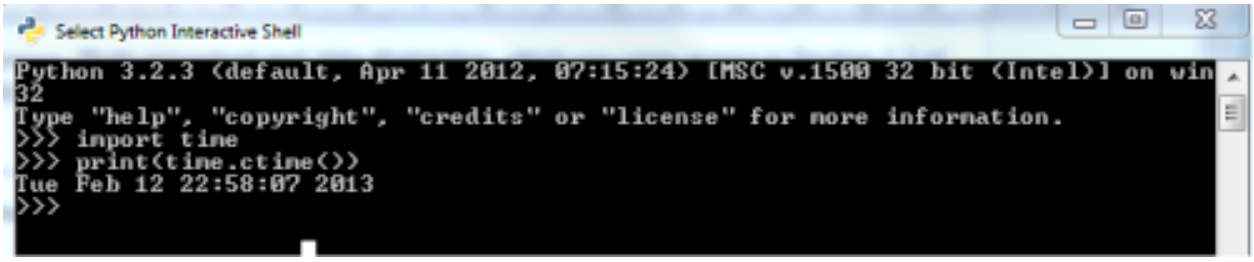


Figure 13 Execution of the small program above in the Python Interactive Shell

How does `time.ctime()` work? We don't know and *we don't need to know*! Someone has written the `time` module and documented its interface, which contains the function `ctime()`. We can just use the function without any knowledge whatsoever of its implementation.

However, under the bonnet the `time` module is itself a program, written in C, another high-level programming language! The programmers of the module, when they wrote it, only needed to know C. They were able to ignore the details of the low-level machine language that the C program would get translated into.

So now we have three layers of programming abstraction (Figure 14).

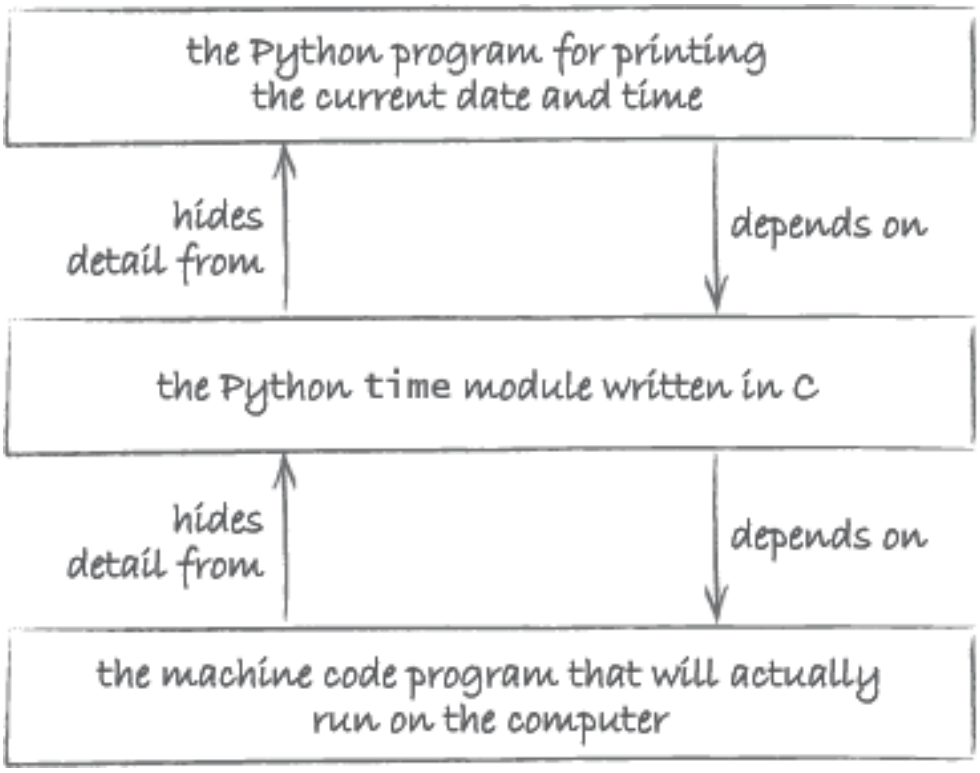


Figure 14 Three layers of abstraction as encapsulation

Think about it: if layers of encapsulation did not exist, the short and easily understood program to display the time (above) would be impossible. We would have to deal directly with instructions written in machine language. There would be many of them and they would be in a language which is very hard for humans to understand.

Programming abstraction doesn't stop there. We can define new abstractions of our own. For example the following function prints the *n* times table up to 9 times *n*."

```
def timesTable(n):
  for m in range(1, 10):
    print(m, 'times', n, 'is', m*n)
```

Once this function is defined we can call it whenever we like without having to bother again with the details.

```
timesTable(9)
```

In fact we can create a Python module that encapsulates this function, so the detail is completely hidden! Suppose we name the module `demo`. Then the `timesTable` function can be used from a different Python program, like this:

```
from demo import timesTable
```

```
timesTable(7)
```

But now we can define a new function that depends on timesTable. The function printUsersTimesTable below prompts the user for a number and then uses timesTable to print the corresponding times table (see Figure 15).

```
Please enter your number: 6
1 times 6 is 6
2 times 6 is 12
3 times 6 is 18
4 times 6 is 24
5 times 6 is 30
6 times 6 is 36
7 times 6 is 42
8 times 6 is 48
9 times 6 is 54
```

Figure 15 The result of the function printUsersTimesTable() when user enters the number 6

```
def printUsersTimesTable():
    aNumber = input('Please enter your number: ')
    timesTable(int(aNumber))
```

So we can build a hierarchy with any number of layers of abstraction, with the detail of each layer hidden from those above and each layer – apart from the bottom one – interacting with the layer below via its interface.

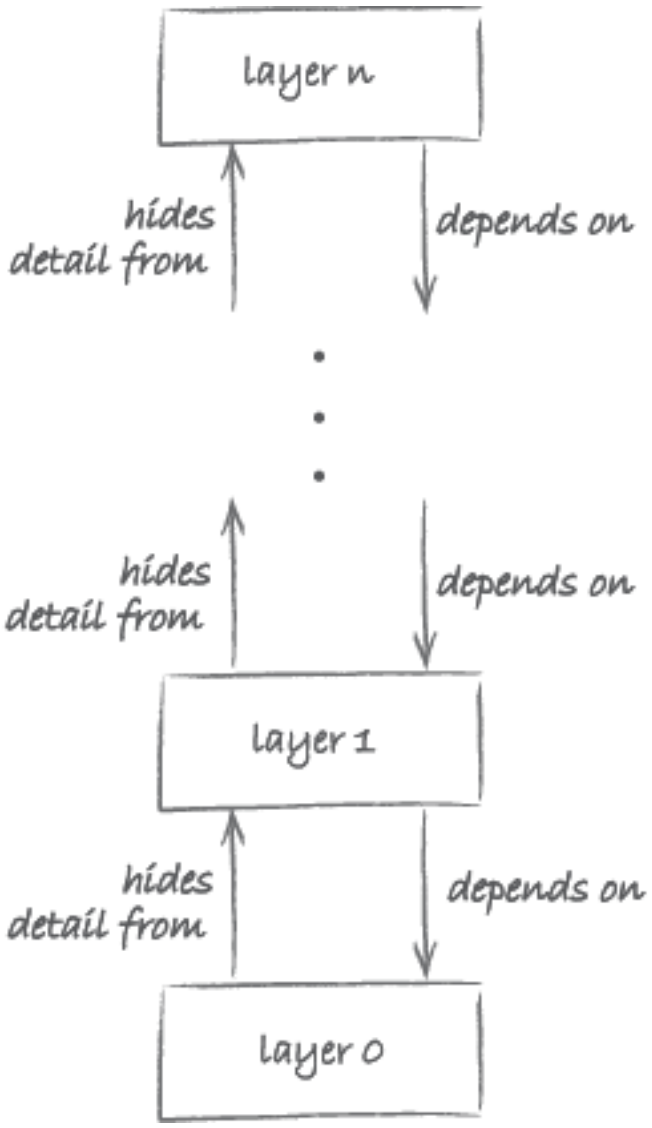


Figure 16 A hierarchy of layers of abstraction (as encapsulation)

The overall picture that emerges is one of a hierarchy of layers. As illustrated in Figure 14, in such a hierarchy Layer 0 hides its details from Layer 1 above it, Layer 1 hides its details from Layer 2, etc. Conversely, Layer 1 depends on Layer 0, Layer 2 depends on Layer 1, etc.

Figure 14 also shows that such a hierarchy eventually bottoms out at Layer 0. That is the layer at which the abstraction is implemented in physical reality, i.e. a substrate/hardware. This physical layer constrains what is possible at the layers that are built on top of it. Nowadays, most computers are built using electronic components. This is, however, by no means the only possible physical substrate. When Charles Babbage (1791–1871) designed the first programmable computer, he came up with a steam-powered machine built of brass and iron. Babbage never finished his machine, but recently part of it has been constructed using Meccano (see Figure 17).

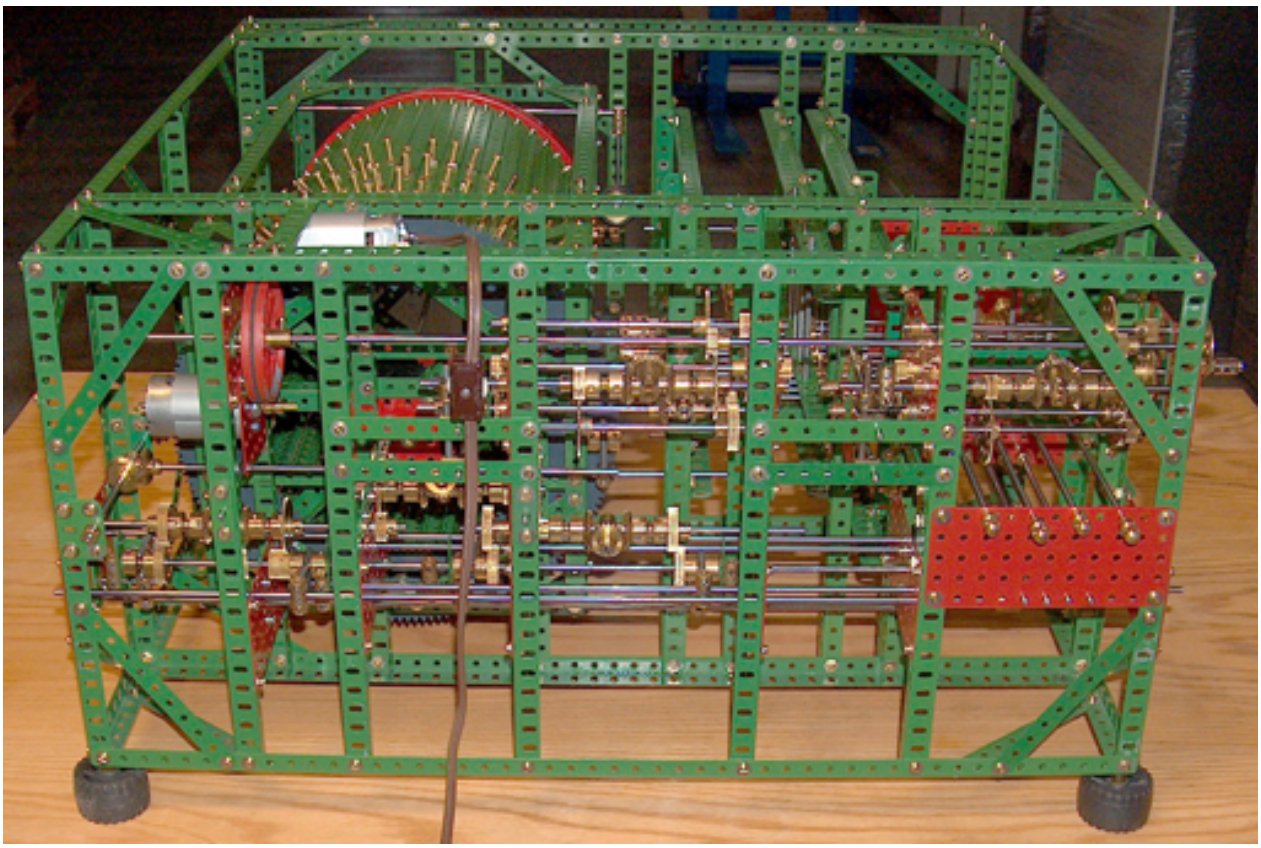


Figure 17 Part of Babbage's Analytical Engine constructed from Meccano

Without going into detail, the idea of layers of abstraction in computing is not restricted to programming languages. It extends, for example, to operating systems and computer networks. In both, a distinction is made between physical and logical layers. Actual devices (e.g. hard drives and computers) and networking hardware (e.g. copper wires) are part of the physical layer. The logical layer contains abstractions (such as IP addresses and protocols).

2.4 Why modelling and encapsulation matter

Both abstraction as modelling and abstraction as encapsulation allow us to manage complexity:

- In the case of modelling, this is done by discarding information: we abstract away all that is irrelevant, to leave a model that contains only what is actually of interest.
- In the case of encapsulation it is done by hiding information: we encapsulate the details of an implementation/automation (of a model) behind an interface.

These strategies are utterly essential to all computer modelling. Without modelling we would be swamped by the complexity of the real world and it would be impossible to define the problem precisely enough to allow it to be solved using an algorithm. Without encapsulation we would be swamped by the detail of the 'computer world' and would find it impossible to deal with the complexity required to implement algorithms as computer programs.

Activity 5 Modelling and encapsulation

In answer to Activity 2 you were asked to construct the diagram in Figure 18. Describe where modelling and encapsulation fit into this diagram.

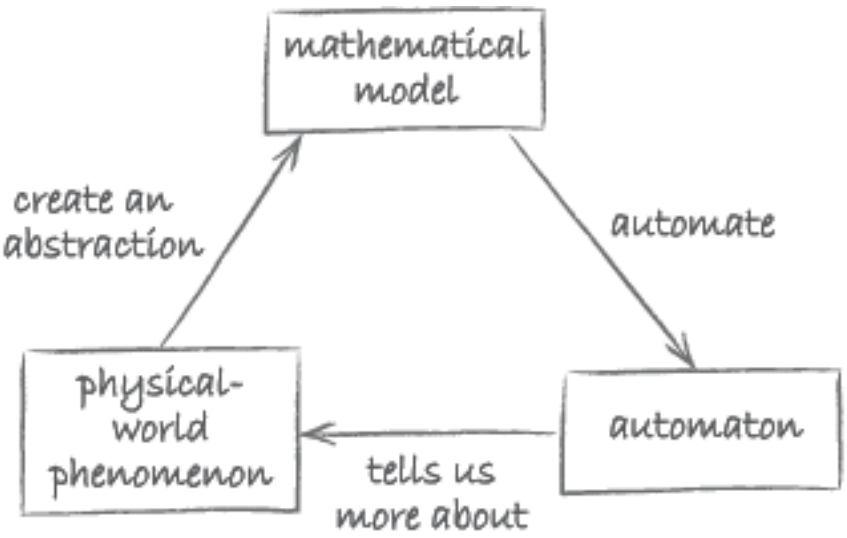


Figure 18 Abstraction and automation

Discussion

Modelling is involved when one creates an abstraction for a physical-world phenomenon. Modelling results in a mathematical model which can then be automated. To deal with the complexity of automating a model, one can use encapsulation.

2.5 Computational thinking: the overview diagram

The aim of this subsection is to draw together some of the concepts you have encountered so far. Consider once more the diagram illustrating some of Wing's ideas (Figure 19).

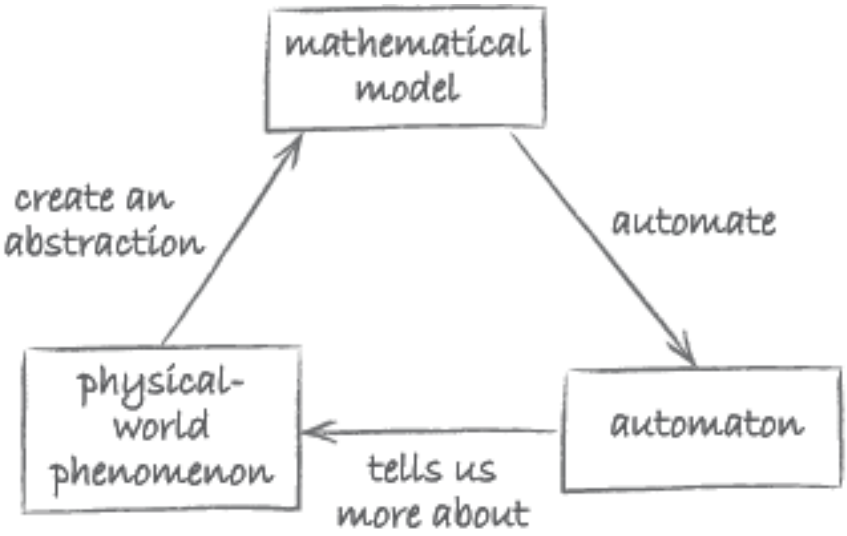


Figure 19 Abstraction and automation

The three main components in the diagram are:

- a physical-world phenomenon
- a mathematical model
- an automaton.

How do these relate to the concepts you met in Section 1, namely:

- real-world problem
- computational problem
- algorithm and data structures?

Let us start with Wing's use of the term 'mathematical model'. As we will now explain, Wing's notion of a mathematical model is linked to that of a computational problem. According to Wing, a mathematical model is an abstraction of the physical-world phenomenon of interest. It ignores irrelevant detail. It also is the point of departure for automation. The fact that it is a *mathematical* model means that it is sufficiently precise to be used as a starting point for creating an automaton. Now, you may recall that we defined a computational problem as a problem that is expressed sufficiently precisely that it is possible to attempt to build an algorithm to solve it. In other words, Wing's mathematical model is nothing other than a computational problem.

Wing's physical-world phenomenon is abstracted by the mathematical model, just like a computational problem provides an abstraction of a real-world problem.

Algorithms and data structures form the link between a computational problem and its automation. Finding algorithms and data structures is part of automating. When these are then implemented as a program and executed, they provide a machine that solves the problem. In Wing's words, the resulting machine – or automaton – automates the abstraction.

The complexity of the machine, as we have just discussed, is managed by having a series of encapsulated layers.

To sum all this up, we can refine the diagram of Figure 19 as shown in Figure 20.

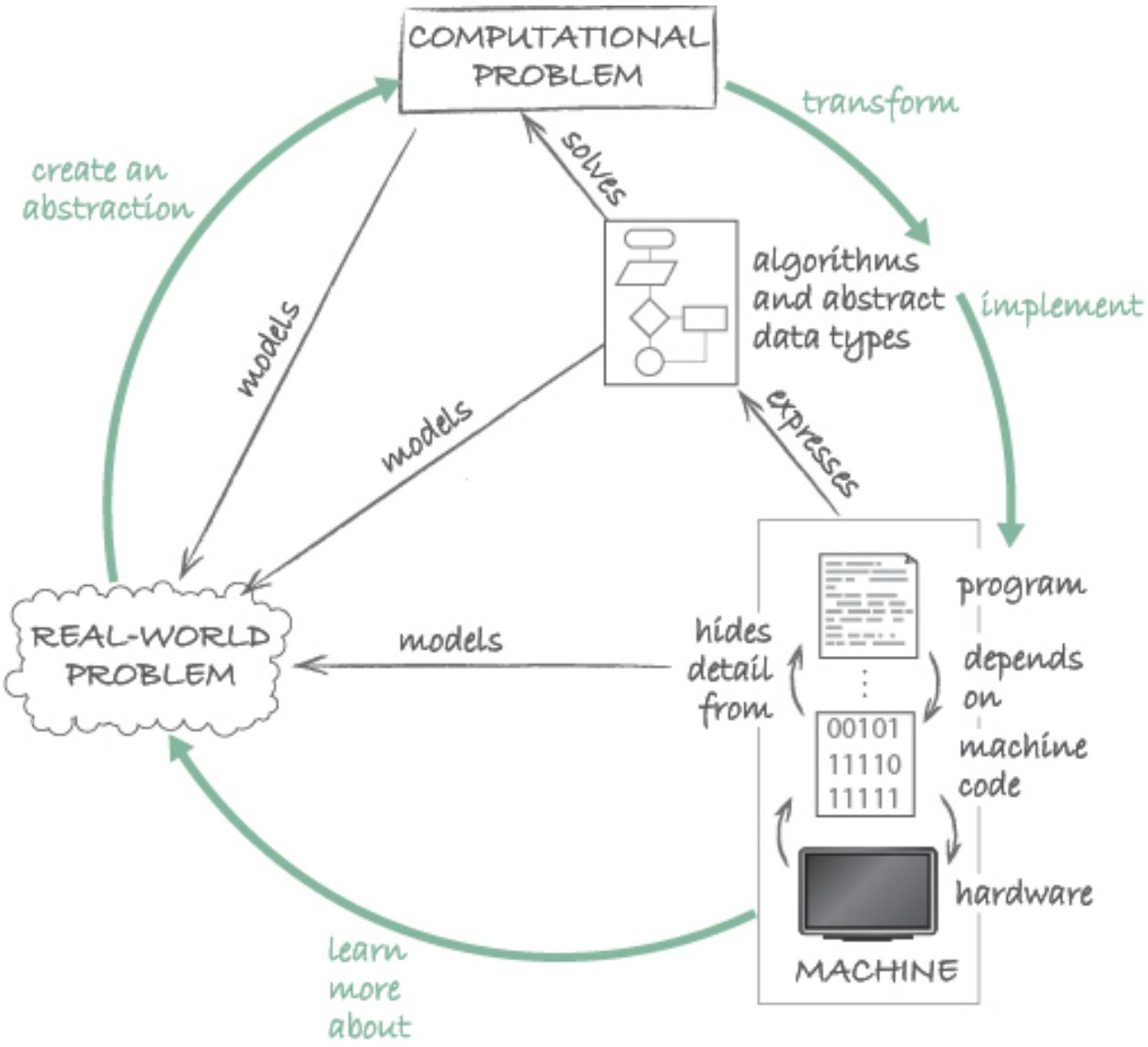


Figure 20 Computational thinking overview diagram

Whereas the core of this diagram represents artefacts (real-world and computational problems, machines) and their relations (models, solves, and expresses), the green arrows indicate the *actions* that a computational thinker engages in (i.e. create, transform, implement and learn).

2.6 Varieties of abstraction

In the next video clip, Wing compares and contrasts abstraction in computational thinking with abstraction in mathematics and engineering. In the following activity you will make these comparisons explicit.

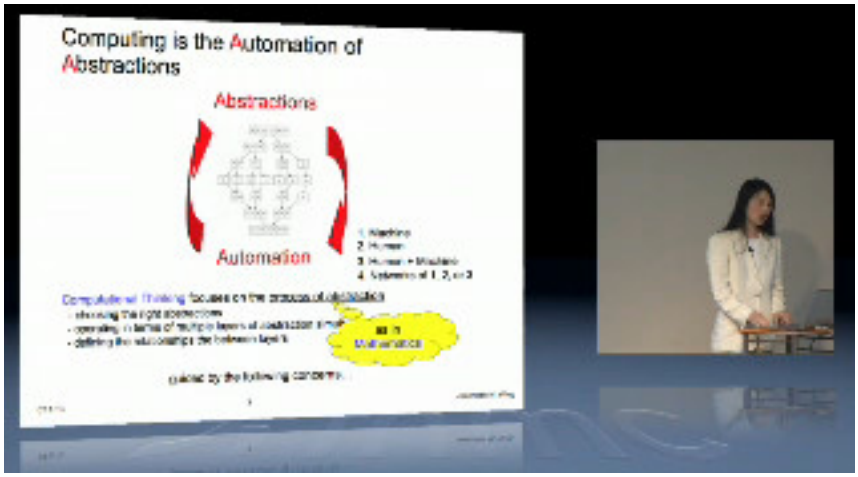
Activity 6 Computational, mathematical and engineering thinking

20 minutes

Parts 1-8

Watch the video and complete the activity by ticking the appropriate boxes.

Video content is not available in this format.
Computational, mathematical and engineering thinking



Part 1

1. Which of the following use formal notation?

- ☐ Computational thinking
- ☐ Mathematical thinking
- ☐ Engineering thinking

Discussion

In all three disciplines, the use of formal/mathematical notation is common:

Part 2

2.Which of the following involve at least two layers?

- ☐ Computational thinking
- ☐ Mathematical thinking
- ☐ Engineering thinking

Discussion

Mathematics typically works only with a single layer: there is no distinction between the ‘properties of interest’ and the ‘ignored detail’. The abstraction is studied in its own right, without reference to some other layer which it represents. Of course, this is an idealisation. Many areas of mathematics did start as a representation of some aspect of the real world. For example, geometry began with Euclidean geometry, which was intended as a model of physical space.

Both engineering and computing rely on at least two layers. At the basis, there is a layer grounded in the real world. This is the layer in which the problem of interest lies (for example a bridge or car in engineering). In addition to this layer, there is at least one further layer which ignores certain of its details. For instance, an engineer may use mechanics to calculate the forces operating on a bridge when a car crosses it. In doing

so, the engineer will abstract away (or ignore), the colour in which the bridge is painted, the brand of the car, and many other things that are not relevant for calculating the forces in question. We've seen that a computer scientist who simulates a dictionary may, similarly, ignore certain properties of a physical dictionary.

Part 3

3. For which of the following are beauty and elegance measures of goodness?

- ☐ Computational thinking
- ☐ Mathematical thinking
- ☐ Engineering thinking

Discussion

In all three disciplines, beauty and elegance are criteria that are used to compare competing abstractions. These are, however, more central in mathematics, where the abstraction is often judged in its own right, without reference to an underlying reality that it represents.

Part 4

4. For which of the following is correctness a measure of goodness?

- ☐ Computational thinking
- ☐ Mathematical thinking
- ☐ Engineering thinking

Discussion

In connection with this criterion, Wing asks in particular: 'Does it do the right thing?' and 'Does it do anything?' These questions are clearly relevant for both computational and engineering abstractions. It is perhaps less clear what they mean in the context of mathematics. Arguably, since mathematics is about the abstraction itself, one can't really distinguish correct from incorrect abstractions because there is nothing to compare them with. However, even when dealing with pure abstractions, we may want to make sure that they are consistent. A mathematical theory that states that a mathematical object both does and does not have some property would be in violation of this criterion (e.g. a theory which says that two lines cross and do not cross each other).

Part 5

5. For which of the following is efficiency a measure of goodness?

- ☐ Computational thinking
- ☐ Mathematical thinking
- ☐ Engineering thinking

Discussion

In connection with computing, Wing mentions three dimensions of efficiency: time (how fast?), space (how much space?) and energy (how much power?). Each of the three efficiency criteria can also be applied in engineering contexts (e.g. how fast can the engine run, how much space does it occupy, and how much energy does it consume?). None of these considerations has a clear equivalent in mathematics.

Part 6

6. For which of the following are usability, modifiability, maintainability and cost measures of goodness?

- ☐ Computational thinking
- ☐ Mathematical thinking
- ☐ Engineering thinking

Discussion

All of these practical considerations are important in both engineering and computing contexts. They are not directly applicable to mathematics.

Part 7

7. Which of the following are constrained by physics?

- ☐ Computational thinking
- ☐ Mathematical thinking
- ☐ Engineering thinking

Discussion

Both engineering and computing are constrained by physical reality. For example, the classical model of computing uses the bit (0 or 1) as the basic unit of information. It is constrained by whichever physical means is used to represent a bit (whether it be an electrical current or, for instance, the position of a mechanical lever). The physical hardware constrains, among other things, how quickly the state of a bit can be changed (from 0 to 1 or vice versa).

Part 8

8. Which of the following can go beyond physics?

- ☐ Computational thinking
- ☐ Mathematical thinking
- ☐ Engineering thinking

Discussion

Mathematics, as the study of pure abstractions, is independent of physical reality. Computing, though always constrained by the physical layer at which the hardware operates, allows us to build on top of the hardware layer software layers that model non-existent realities. By automating these virtual realities, it allows us to see and even experience realities that go beyond what is physically possible.

2.7 Virtual worlds

Though engineering and computing have much in common, Wing argues that one big difference between the two is that computing allows us to escape from the constraints of the physical world. First, however, she points out that automatons (i.e. entities that carry out automation) are constrained by physics in the sense that they require a physical substrate/hardware to exist and work. In the end, any algorithm can only run on a real machine – a machine made of hardware – whether it comprises electronic components, brass and iron, or brain tissue.

Let us pursue the idea of how computing allows us to step outside the realm of physical constraints. For that we need to revisit the idea of abstraction as modelling. You have seen that algorithms can work with abstractions of real things such as paper and ink dictionaries. Perhaps more spectacularly, algorithms can also work with abstractions of things that do not exist, at least not in the physical world around us.

Activity 7 Abstractions of the non-physical

Can you give an example of an algorithm that works with abstractions of non-physical things? Hint: You can find such an example in Section 1 of this course.

Discussion

You saw an example in Section 1 when you learned about the algorithm for column addition. This algorithm works with non-negative integers. At first sight, it might seem that integers are physical objects. After all, can't we write them down using pen and paper? Recall, however, Activity 4 (Ceci n'est pas une pipe). In that activity you studied the distinction between representations/abstractions and the reality they model. When you write down a '2' (or any other number) on a piece of paper, this is a representation of the number 2, not the actual number. Why? Suppose that what you wrote down was the number 2 itself. In that case, if you erased it, 2 would cease to exist, and if you copied it, you would have created another number 2. That would be absurd! When you write down a '2', this is merely a representation, not the actual number. Other people can write down a '2' as well, and thereby talk about the *same* number that you're talking about when using a '2'.

Perhaps this has set you wondering what numbers are, if they aren't physical objects. Unfortunately, philosophers have been at loggerheads about this question since the ancient Greeks started discussing it. Many mathematicians subscribe to a view championed by the philosopher Plato (429–347 BCE), who proposed that numbers exist, but not in space and time; rather they inhabit what is known as 'platonic heaven'.

Wing gives a further example of abstractions in computing that model something (physically) non-existent. She discusses virtual worlds, such as Second Life, a three-dimensional virtual world populated by avatars. Avatars are computer-animated characters that are controlled by real people – computer users. The avatars can do many things that we are familiar with in the real world, including walking around, talking to people, and interacting with objects. They can, however, also do things that are not possible in the real world, such as teleport from one location to a geographically distant other location. This is another instance where the automation of abstractions allows us to go beyond the things that can exist in the physical world. Note the big difference here between computing and conventional engineering. Engineers are generally interested in creating artefacts that are physically possible – what would be the point of designing a bridge that cannot be built?

3 Computational thinking everywhere

At the very beginning of this course, computational thinking was presented as a desirable skill – arguably the skill for the twenty-first century. Why? Well, in recent years, computational thinking has led to exciting developments and discoveries in a number of

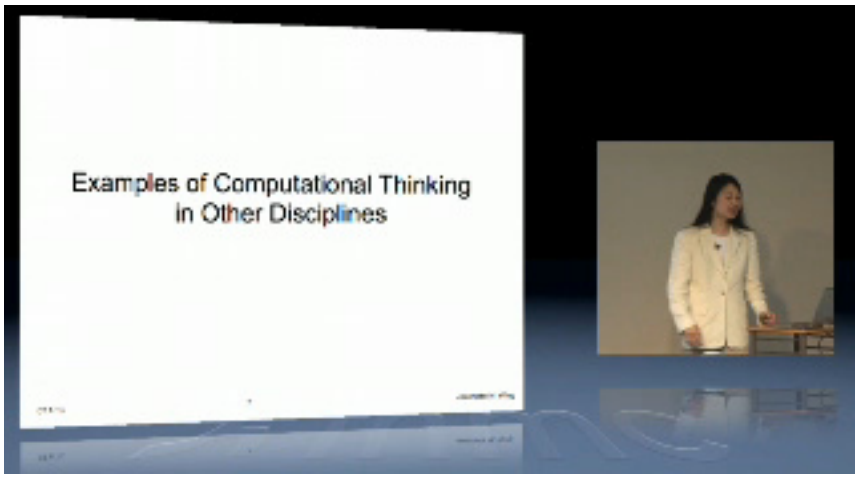
fields. This has caused a demand for people with computational thinking skills to move these fields forward even further. In this section, you will encounter some of these applications of computational thinking. Watch the videos, but don't worry if you can't follow all of it. We want you to get a feel for the big picture – you're not expected to remember everything in detail.

Activity 8 Computational biology

4 minutes

Watch the following video.

Video content is not available in this format.
Computational biology

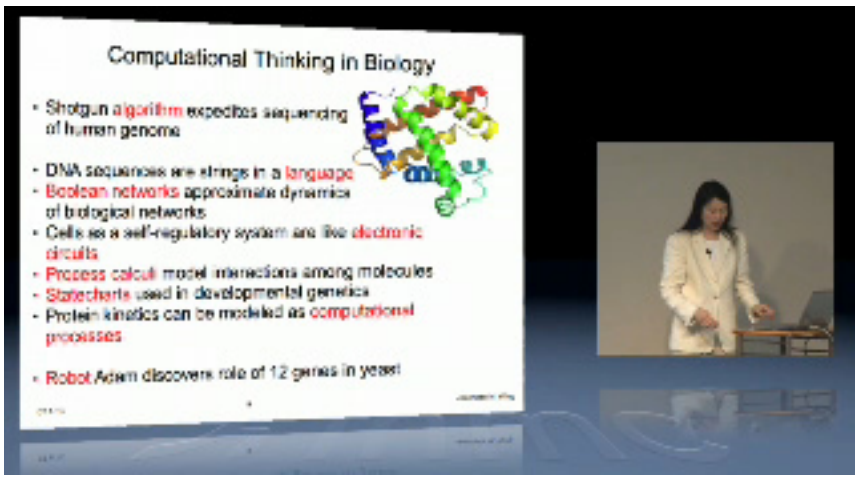


Activity 9 Model checking

8 minutes

Watch the following video.

Video content is not available in this format.
Model checking



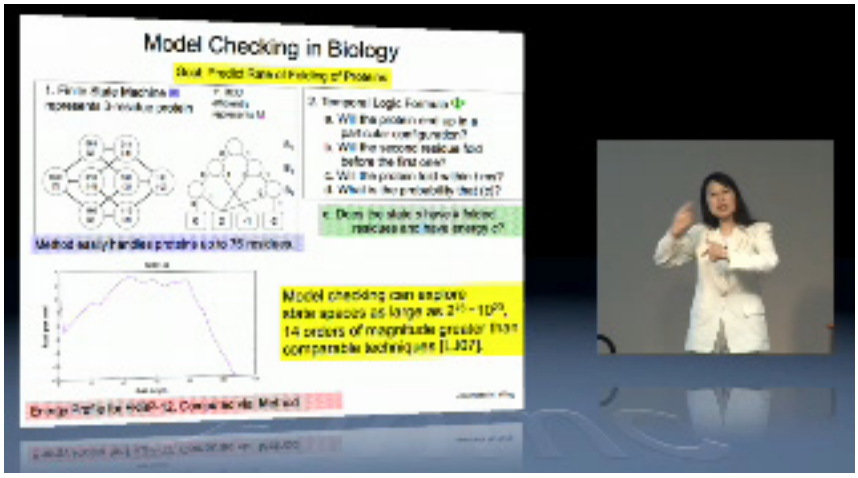
3.1 Machine learning

Machine learning is a technique for automatically finding patterns in large amounts of data. Watch the following video extract in which Wing discusses several applications of machine learning.

Activity 10 Machine learning

15 minutes

Video content is not available in this format.
Machine learning



Wing gives the example of basketball coaches who use machine learning to find out which skills distinguish good players. Once they know this, they can teach those skills to their own players. Machine learning makes this possible by automatically finding patterns in the behaviour of professional players from a large collection of video recordings.

Describe this use of machine learning in terms of abstraction as modelling.

Discussion

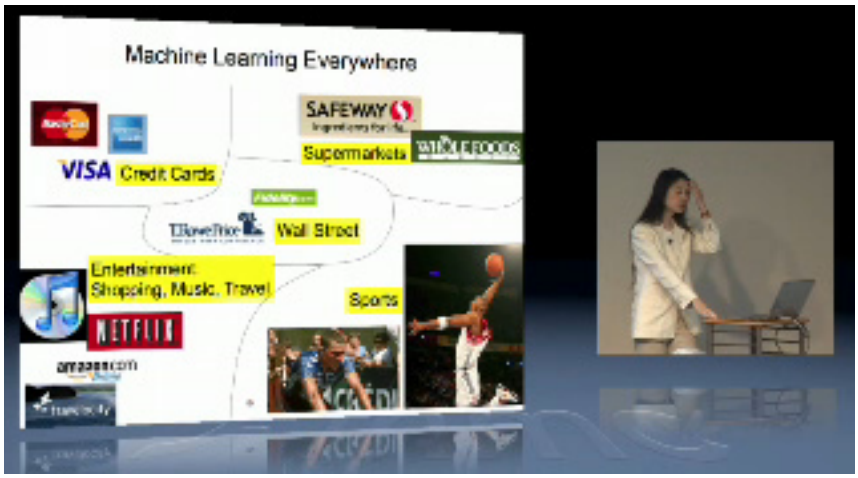
One can view this use of machine learning as an attempt to automatically build a model of a good basketball player. Such a model is an abstraction that needs to capture the skills that distinguish a good player and ignores anything else. At a different level, machine learning is itself based on a model of learning by humans or animals. Being an abstraction, it ignores many of the details of human and animal learning, while preserving some key properties (in particular, the idea that one can learn from examples).

Activity 11 Chemistry, physics, economics …

6 minutes

Watch the following video.

Video content is not available in this format.
Chemistry, physics, economics...



Conclusion

The underlying theme of this course has been ‘computational thinking’. We’ve defined this as consisting of the skills to:

- formulate a problem as a computational problem
- construct a good computational solution (an algorithm) for the problem, or explain why there is no such solution.

We introduced the idea of an algorithm with two examples: an algorithm for addition and one for dictionary search. You learned about the concept of abstraction and its two varieties: abstraction as modelling and abstraction as encapsulation. We contrasted and

compared computational thinking with both mathematical and engineering thinking. You have also been shown the 'bigger picture': the way that computational thinking is shaping research in many disciplines, ranging from biology and physics to economics and sport science.

Keep on learning

Study another free course

There are more than **800 courses on OpenLearn** for you to choose from on a range of subjects.

Find out more about all our free courses.

Take your studies further

Find out more about studying with The Open University by visiting our online prospectus.

If you are new to university study, you may be interested in our Access Courses or Certificates.

What's new from OpenLearn?

Sign up to our newsletter or view a sample.

For reference, full URLs to pages listed above:

OpenLearn – www.open.edu/openlearn/free-courses

Visiting our online prospectus – www.open.ac.uk/courses

Access Courses – www.open.ac.uk/courses/do-it/access

Certificates – www.open.ac.uk/courses/certificates-he

Newsletter – www.open.edu/openlearn/about-openlearn/subscribe-the-openlearn-newsletter

Glossary

abstraction

provides a handle on complexity by either ignoring detail by means of a **model**, or hiding detail through the use of **encapsulation**.

algorithm

A precisely stated, step-by-step list of instructions.

data structure

A way of organising data for use by an **algorithm**.

encapsulation

Hiding details of an implementation from users. This is achieved through an interface that sits between the user (or client) and the implementation.

interface

An interface sits between a user (or client) and an implementation. It hides the details of the implementation from the users or clients of the implementation. (See encapsulation.)

model

A representation or **abstraction** of a part of reality which ignores certain details.

simulation

A working **model** of a process or a system which changes over time.

References

Gough, J. (1767) *Practical Arithmetick in Four books*, Dublin.

Miller, B. N. and Ranum, D. L. (2011) *Problem Solving with Algorithms and Data Structures using Python*, 2nd edn, Sherwood, Oregon, Frankin, Beedle & Associates.

Wing, J. M. (2006) 'Computational Thinking', *Communications of the ACM*, vol. 49, no. 3, pp. 33–5; also available online at http://www.cs.cmu.edu/~wing/publications/Wing06.pdf (Accessed 20 December 2012).

Wing, J. M. (2009) 'Computational Thinking and Thinking About Computing', *Evening Lecture Series*, 2 September 2009, Florida Institute for Human and Machine Cognition, Florida, US [Online]. Available at http://www.youtube.com/watch?v=C2Pq4N-iE4I (Accessed 20 December 2012).

Acknowledgements

This course was written by Paul Piwek (with input from Alistair Willis for Activities 1 and 3).