The Magic of Computer Science III

Magic meets mistakes, machines and medicine

Presented by
Paul Curzon
and Peter McOwan
Queen Mary University of London

A Computer Science for Fun / Teaching London Computing Special

Queen Mary
University of London

Contents

The Magic of Machines Saving Lives

Magic, Medicine and Computational Thinking

Expert programmers are often called wizards. They use their skills to work magic in creating computer programs that do all the amazing things computers do. It turns out, programmers really do use a lot of the same skills as magicians: computational thinking skills. When those computers are designed to help medics in hospitals, it's about the magic of saving lives.

This book contains magic tricks showing you what computational thinking is all about. Along the way we will also see how mathematics, psychology, design and the social sciences matter to both magicians and computer scientists, and so ultimately to medicine too.

Keep the magician's code

Some of these effects are in the shows of professional magicians. We present them here for educational and entertainment purposes. If you perform them for friends don't break the magicians' code. Practice the tricks first and never reveal the secrets.


Invisible
Palming

Invisible Palming: Magic

You invisibly move a card from one pile to another. A volunteer can do it too, even though they have no idea how.

The mechanics

Take 15 similar looking cards from a shuffled pack – all black, no royalty or Aces. Have a volunteer put their hands with fingers and thumbs touching the table as though playing the piano. Explain that everyone must chant the magic words: "Two cards make a pair".

Take two cards and as everyone says "Two cards make a pair" place them together between a pair of fingers. Keep doing this until you have one card left. Place it between the final fingers saying there is "one left over".

Now take the first pair back, again all chanting "Two cards make a pair". Place them face down on the table to start two piles. Do this with each pair: saying the magic words and adding one card neatly to each pile. Eventually only the last single card is left. Take this card saying: "We have one extra card." Let them place it on top of one of the piles. Square up the piles pointing out: "So that pile now has the extra card".

Explain that you are going to do 'Invisible Palming'. The extra card is on one pile. You are going to invisibly move it to the other. Place your hand over the pile with the extra card. Rub the back of your hand to "make the card go invisible". Lift your palm showing that the card you are pretending to move is invisible. Move your hand to the other pile. Tap it, "to make the card drop". Announce that the card has now moved piles. (In fact you did nothing at all!)

To show the magic worked, take the pile where the extra card was placed and count off pairs into a new single face down pile – "Two cards make a pair. Two cards make a pair…". This pile must be neat so no one counts the cards. You find there are only pairs – the extra card has disappeared! So where has it gone? Take the other pile and do the same, putting pairs back into a pile. Amazingly the extra card is there. Exclaim that the extra card really has moved from one pile to the other!

Now tell the volunteer that they can do the trick. Put your hands out in the piano position and talk them through the steps. To their surprise they manage to move the card, even though they don't know how.

How it works

Magicians call this a self-working trick. It always works if you follow the steps. It appears magical because you have confused everyone. They believe when they add the last card to a pile they are adding an extra odd card. You are actually making up the last pair – making an odd pile even. There are 15 cards. After dealing out the pairs there are 7 cards in both piles. The last card makes its pile up to 8 cards – 4 pairs. When you count out the pairs there will be only pairs there, so no 'extra' card. The other pile will be left with 7 cards: 3 pairs with one left over. You pretend it has magically moved without doing anything. Nothing has to move!

Invisible Palming: Computational Thinking

Algorithms and Magic

What does this have to do with computing? Well, Computer scientists call self-working tricks algorithms. An algorithm is a series of instructions that if followed exactly and in the right order lead to a guaranteed effect. The instructions have to be precise and cover all eventualities. For example, this trick has to work however the cards are shuffled and wherever the extra card is placed.

The Invisible Palming Algorithm

Here it is written out as a series of steps.

To do Invisible Palming:

1) A volunteer places their hands as though playing the piano.

2) Do the following 7 times.
   a. Place 2 cards between two of the volunteer's fingers.
   b. Say "Two cards make a pair".

3) Place a single card between the remaining finger and thumb.

4) Do the following 7 times.
   a. Take a pair from the fingers and split them in two piles.
   b. Say "Two cards make a pair".

5) Ask the volunteer to place the last card on a pile of their choice.

6) Place your hand over that pile, rubbing the back of your hand.

7) Lift your hand, show the palm then place it over the other pile.

8) Tap your hand and remove it, saying the card has moved.

9) Pick up the first pile.

10) Do the following 4 times.
    a. Add 2 cards from the pile you are holding to a pile on the table.
    b. Say "Two cards make a pair".

11) Pick up the second pile

12) Do the following 3 times.
    a. Add 2 cards from the pile you are holding to a pile on the table.
    b. Say "Two cards make a pair".

13) Reveal that the extra card is now in the second pile.

Invisible Palming: Computational Thinking

Deviously Decomposing

To do the trick we start at the first instruction and work through them in turn. Magic tricks have to be written in enough detail that a budding magician can follow the instructions and get the magical effect.

That is the point of algorithms. Once you have an algorithm that works, you don't have to think about it any more. You don't have to solve the problem of how to get the effect. You just blindly follow the algorithm and it will happen. The trick shows this. The volunteer following your instructions gets the magical effect even though they have no idea how it works.

The detail matters. Just saying we should pretend to move a card from one pile to the other isn't good enough. We need to say how to do it. However, we can split a trick into separate parts. Spinning off separate algorithms to do smaller parts of the task makes it all easier to understand, while still giving all the detail. It's called Decomposition.

The Trick Decomposed

Here is a new description of the trick using decomposition. We give all the detail for each step. Each is an algorithm of its own.

To do Invisible Palming:

1. Deal out 15 cards in pairs with one left over.
2. Take the pairs back splitting them into two piles.
3. Have a volunteer choose where to put the extra card.
4. Pretend to move a card between piles
5. Reveal that the extra card has magically moved piles.

To deal out 15 cards in pairs with one left over:

1. A volunteer places their hands as though playing the piano.
2. Do the following 7 times
   a. Place 2 cards between two of the volunteer's fingers.
   b. Say "Two cards make a pair"
3. Place a single card between the remaining finger and thumb.

To take the pairs back splitting them into two piles:

1. Do the following 7 times
   a. Take a pair from the fingers and split them in two piles
   b. Say "Two cards make a pair"

To pretend to move a card between piles:

1. Place hand over pile with last card rubbing the back of the hand
2. Lift your hand, showing the palm and placing it over the other pile.
3. Tap your hand and remove it, saying the card has moved.

To reveal that the extra card has magically moved piles:

1. Pick up the first pile.
2. Put 4 pairs down
3. Pick up the second pile
4. Put 3 pairs down
5. Reveal that the extra card is now in the second pile

To put n pairs down:

1. Do the following n times.
   a. Add 2 cards from the pile you are holding, to a pile on the table.
   b. Say "Two cards make a pair".

Invisible Pairing: Computational Thinking

Algorithmic Thinking

Computer programs are algorithms too. Computers work by following the instructions written by programmers. Magic tricks are written in English so a human can follow them, programs are written in a programming language so a machine can follow them blindly.

Programs contain similar structures to our magic trick's description. They are a sequence of instructions to be done in order. They have instructions that say other instructions are to be repeated (**loops**). They are split into separate named parts (**procedures**), each an algorithm for a simpler task. To a computer scientist:

To Deal out 15 cards in pairs with one left over: is a **procedure**. Some procedures need to be given information to do their job (called **parameters**). For our trick n was a parameter in the procedure "Put n pairs down". This allows it to be used both to explain how to put 4 pairs down (with n as 4), then again to put 3 pairs down (with n as 3). One procedure does both jobs.

A magician who invents new tricks is creating algorithms. A programmer writing a program is doing the same. They are both using their computational thinking skills and especially **algorithmic thinking**. It involves working out a series of steps that will always have the effect you are after. Algorithmic thinking also involves writing instructions really clearly and precisely so that there is no confusion about what to do.

Programmers really are wizards!

Invisible Palming: Medicine

Algorithms in Hospitals

Sarah is very ill in hospital intensive care. To get better she needs a constant supply of medicine. A computer gives it to her. A syringe full of the drug has tubes that go into the veins in her arm. The computer controls a pump that pushes the syringe, gradually giving her the medicine she needs.

An algorithm is making Sarah better. A programmer wrote the instructions telling the computer how to control the pump. It is following an algorithm just like a magician following the steps of a trick. The algorithm first allows the nurse to enter the amount of drug to be given. It then waits until the nurse presses the start button. Then it switches on the pump at the right speed to deliver the right amount of drug. Finally, it stops the pump when the total dose had been delivered, having calculated the time the pump should run for. It deals with lots of other things too – presenting suitable information on the screen, sounding alarms if the tubes get blocked, and so on.

Hospitals are full of computers disguised as medical devices. They monitor patients, run tests to help work out what is wrong, and give treatments, from medicines to radiation therapy. They even help with surgery. All are following algorithms written by computer scientists.

It is not only Doctors and Nurses keeping patients alive. The engineers who design the machines and the computer scientists who program them do too. They all work together.

Ninja Countdown

Ninja Countdown: Magic

You make two piles of cards that a volunteer quickly covers with their hands. Somehow you still move a card from one pile to another.

The mechanics

This trick has the same ultimate effect as Invisible Palming. You appear to move a card from one pile to another. It has completely different mechanics though.

Take 12 cards from a normal pack and count them on to the table face down. Note there are 12 cards. Then count them again just to double-check, but counting down this time 12, 11, 10, 9, etc.

Explain you are going to split them into two piles, then use your Ninja powers to move a card from one pile to the other. As you are lightning quick, when you get to 6 your volunteer should slam their hand on the pile as quickly as they can. They must beat you, as you will be trying to take a card. Count cards on to the table starting from 12 till you get to 6 ("8,7,6 in that pile") and let them slam their hand down. Then say "and the other 6 go in a second pile. Slam your hand on them too as soon as I put them down".

Say they were too fast so instead you will need to do it using your lightning fast ninja powers despite their hand being there. Have some fun pretending to remove a card from the second pile adding it to the first.

Remind them that there were 12 cards and you counted 6 cards in to the first pile and 6 in to the second. Tell them to remove their hands and count each pile. They find there are now only 5 cards in one pile, but 7 in the other. You moved a card with your amazing Ninja skills.

How it works

This trick again works by confusing people into thinking a card has moved when it was in the 'new' pile from the start. Here people get confused about counting down. When you count the cards, first counting up and then counting down, you are showing that counting down works the same as counting up. But actually it doesn't unless you count all the way to 1. When you count out the first pile and stop at the halfway number (here 6) you have actually counted one more than 6. Counting 12, 11, 10, 9, 8, 7, 6 gives you 7 cards in that pile not 6! So in plain sight you have just counted an extra card onto that pile, leaving one less on the other pile.

Ninja Countdown: Design

Entering Numbers

Tricks like this show that anyone can be confused over numbers even over simple things like counting backwards. Numbers are everywhere. We enter them to set alarm clocks, microwaves and the central heating, to get money, use a credit card and set the house alarm. Not getting confused over numbers matters.

In a hospital numbers are entered in to machines all the time: volumes, times, rates, weights, ages, patient IDs and more. Doctors and nurses can't afford to get confused. We are easily confused by small differences to what we are used to, and yet different wards often have different machines to do the same thing. Different models expect numbers to be entered in different ways, by pressing different sequences of buttons, for example, even on otherwise identical looking devices. Each manufacturer has its own way of doing things. Often the manuals assume it is obvious how to enter a number, or say misleading things like "numbers are entered just like on a calculator" when different calculators have different ways of entering numbers. Even something as simple as the layout of keys changes – compare mobile phones with calculators and other gadgets. A nurse who enters numbers all the time in one ward has to be extra careful when using machines somewhere else. Developing clear and accurate instruction manuals is a really important part of developing software, but better still the designs need to help.

Ninja Countdown: Design

Standard practice

One solution is standardisation. Rather than having a manufacturer free-for-all, we set up prescribed ways of doing things – Standards – with regulators to check they are followed.

Standards make all our lives easier. They tell us how to refer to different currencies, ensure food labelling is consistent and make sure plugs of all electronic gadgets fit any socket, for example. In the long run standards are better for everyone including manufacturers.

Standardisation fits well with the computational thinking idea of **generalisation**. Once a computer scientist has solved a problem they try and make their solution as general as possible and package it up so it can be used for lots of other problems too. Rather than reinventing new solutions you can then just take a ready packaged one you prepared earlier. Standardisation is doing a similar thing.

Confused connections

Before you standardise though you need to know the best way of doing things. You need a good design to standardise! It's not always simple. For example, it might seem sensible to standardise all the connectors that connect the hospital tubing used to pump fluids into people – blood, drugs, nutrients and so on. If they are all the same then they will be easy to set up for a new patient. However, that would lead to accidents due to mix-ups. You don't want to accidentally connect the tubes into someone's spine that should deliver the drug into their arm. By designing different connectors you can make it impossible to get it wrong: by design. Standards can be about standardising good differences as well as about making things the same.

Standards are still lacking for entering numbers on medical devices. In medical situations it really matters so researchers are now working on the best ways to do it, so that in future it will be easier for nurses. Good design can, as we will see, help make sure nurses don't enter the wrong numbers and help them notice when they do.

The Magical Friendship Test

The Magical Friendship Test: Magic

This magical test checks if two people will be friends for life. Each picks a card in secret. One does a magical calculation and writes the answer for all to see. Amazingly, when the two chosen cards are revealed they give the same number.

The mechanics

Pick a pair of friends from the audience. Give the Ace to 9 of Hearts to one and those of Spades to the other. Ask each to shuffle their cards, pick a card without seeing it and put it face down on the table. You need to see the chosen Spade. To do this get them to each look at the card they chose and remember it. Stand with the Spade person and helpfully show what you want them to do next by gathering up the face down spades from the table into a loose pile in your hands. Get them to return the chosen spade to the bottom of this pile, square up the cards by rotating them with both hands to tidy the pack and push all cards in. Distract them by asking the other person to do the same. As you do this tilt the pack up slightly and catch a glimpse of the bottom card. Hand the spades back. Place both piles face down side-by-side on the table, chosen cards at the bottom.

Friendship Compatibility Number

Next talk the person with the Hearts (whose card you didn't see) through calculating the pair's 'Magical Friendship Compatibility Number'. Suppose the first person picked the 3 of Hearts (though you don't know this) and you saw that the other person had the 8 of Spades.

On a piece of paper they do the following calculation:

Double the number they chose. (3 x 2 = 6)

Add 2 to the result. (6 + 2 = 8)

Multiply the total by 5. (8 x 5 = 40)

Subtract a **magic number you give them:** 2. (40 - 2 = 38)

Finally they write the result, here 38, for all to see.

You point out that the closer the result is to their chosen cards the more compatible they are. The cards were picked randomly. No one saw both, so no one knew what number they would make. Turn over the two piles revealing the chosen cards, and place them side by side, Heart (3) then Spade (8). They magically give exactly the same 2-digit number (38) as was written down! The test has shown the pair will be friends for life...

Making it work

It worked with those cards, but how do we make it always work? The trick is in the last magic number subtracted. It should always be the spade's value you saw subtracted from 10. We saw the 8 of Spades, so worked out 10 - 8 = 2. So that time 2 was the magic number. If they had picked the Ace then you do 10 - 1 = 9 in your head and 9 is the magic number you tell them to subtract. Calculate the right magic number like this and it always works.

The Magical Friendship Test: Computational Thinking

A Book of Runes

So we've got a new trick. Better write it down before we forget it. Let's devise a magical notation to do it – Runes.

Name it!

First we need to give names to the different numbers like those written down, and on chosen cards. We need names for the chosen Heart and Spade cards. Let's call them `heart` and `spade`. We need another for the "magic number" we calculate from the Spade. Let's call that `magic`. Next we need names for the different totals as the volunteer works through the calculation: `total1`, `total2`, `total3` and then `friends` for the 'friendship' number that everyone sees. Finally, we need the number revealed by putting the two chosen cards together. Let's call that `reveal`.

Runic Commands

We also need a way to write down what calculations need to be done and in what order. We will use a runic arrow: <~ to mean link the result of a calculation with a name. The & symbol will be runic punctuation for the start and end of an instruction. For example, the instruction saying how to get the magic number is:

```
& magic <~ 10 - spade &
```

It just means, subtract the number on the Spade card from 10 and name the answer `magic`. We could add instructions for what the Magician says too but let's stick to calculations for now to keep it simple.

Choose me!

We need a runic command for choosing cards. Let's invent a command `CHOOSETH` that is followed by the numbers to choose from. It says randomly choose a value. The value is given a name using <~ as before. We do that twice before calculating the magic number:

```
& spade <~CHOOSETH{1,2,3,4,5,6,7,8,9}&

& heart <~CHOOSETH{1,2,3,4,5,6,7,8,9}&
```

The next calculation is to take the number on the Heart card and double it, naming the answer, `total1` i.e., write it on the first line of the paper. We will use x to mean multiply.

```
& total1 <~ heart x 2 &
```

Next we take `total1`, add 2 to it and write the new number down on the next line, which we call `total2`. Then we multiply the number `total2` by 5 to give `total3`. Lastly, we subtract the magic number from `total3`, to get the final friendship number we call friends.

```
& total2 <~ total1 + 2 &

& total3 <~ total2 x 5 &

& friends <~ total3 - magic &
```

The Magical Friendship Test: Computational Thinking

Reveal the truth

We need to reveal the 2-digit number given by putting the chosen cards side by side. We will use ^ to mean do that, so for example 5 ^ 4 is 54 and 10 ^ 12 is 1012. We had 3 ^ 8 which became 38. We call the resulting number reveal.

```
& reveal <~ heart ^ spade &
```

The last step involves checking that `reveal` and `friends` are identical and if so say that they will be friends forever. Let's have a new kind of runic command for this, `EQUALETH`, that checks if two things are the same. To show it is a test we will put it in runic curly brackets. We want to check if reveal and friends are the same so our test is:

```
{ reveal EQUALETH friends }
```

We want to take an action based on the test. If (and only if) the two numbers are equal, we want to say that they will be friends for life. Let's just treat speaking like writing numbers. We will use the same symbols, but have a special name. Let's carry on being a bit Shakespearian and use `SPEAKETH` as the name of information to be spoken.

If a disaster occurs and the trick goes wrong we will need to say something, perhaps saying they will fall out instead. We hope it will always work and we will never say that but it's always good to deal with things going wrong.

Back to our instructions. Let's use the words `IF, THEN` and `OTH'RWISE` to combine a test with the two things to do depending on whether the test is true or false. For our trick we write:

```
& IF { reveal EQUALETH friends }
THEN
& SPEAKETH <~
 "Your friendship will last for ever" &
OTH'RWISE
& SPEAKETH <~
 "Oh no, you are soon to fall out" &
```

Putting everything in the right order, we get our full rune (see box). Anytime we want to do the trick we consult the rune.

```
TO DOETH: The Friendship Test:
& spade <~CHOOSETH{1,2,3,4,5,6,7,8,9}&
& heart <~CHOOSETH{1,2,3,4,5,6,7,8,9}&
& magic <~10 - spade &
& total1 <~heart x 2 &
& total2 <~total1 + 2 &
& total3 <~total2 x 5 &
& friends <~total3 - magic &
& reveal <~heart ^ spade &
& IF {reveal EQUALETH friends}
THEN
& SPEAKETH <~
 "Your friendship will last for ever" &
OTH'RWISE
& SPEAKETH <~
 "Oh no, you are soon to fall out" &
&
```

The Magical Friendship Test: Computational Thinking

Magic, Runes and Programs

Our rune tells the Magician what calculations to do, what to do with the answers and crucially, the order to do them in. It does it in a very precise language. Guess what! We just invented a programming language and wrote a program. That is all code is, instructions (so algorithms) written in a precise language (a programming language), where every statement has a very precise meaning.

Each basic instruction is an **assignment**: it does a calculation then names the answer. It **assigns** a number to a **variable**. A variable is just a name that refers to a value. More generally, variables are places where numbers are stored that will be needed later. In our rune the places are the paper, the cards and in our heads. When a computer executes a program the variables are storage places in its memory.

Languages for good

The invention of programming languages was massively important. Originally, computers were programmed by writing long lists of numbers. Each number was an instruction. Writing programs like this was really hard to get right. Eventually, along came high-level languages like our runic language, where each instruction was a big step using English like words. It's languages like this that has allowed really big programs to be written. Think back to all those machines in hospitals. Their programs contain perhaps millions of instructions. Programs that big just cannot be written without high-level programming languages. Without them doctors and nurses would be keeping people alive without computers to help.

When things go wrong

As with magic, when writing programs you must plan what the program should do if things go wrong as well as when they go right. We expect our trick to always work, but we still included an instruction of what to do if it doesn't. It might go wrong because there is something wrong with the trick – perhaps it doesn't always work. It might also go wrong because of people doing the wrong thing. In our trick the volunteer might get the calculation wrong. Our instructions cover that. If you are writing a program for a medical device making sure the program can cope when things don't go as expected matters a lot. Nurses often have to do calculations of drug doses before entering numbers. They will sometimes make mistakes and the programs need to help spot them and do sensible things. As a programmer, you must make sure your program ALWAYS does something safe and sensible.

The Magical Friendship Test: Computational Thinking

Testing Times

Human mistakes aside, it would be nice to be sure our trick always works. After all it would be embarrassingly un-magical to announce that the friends will fall out. Even if we can rescue the trick, we don't want to have to. Evaluating algorithms is a core part of computational thinking.

We could just do the trick a few times. Programmers call this **testing** and it's very important. But how many times do you need to test a magic trick to be sure it always works? How much testing would you do before you would do it in front of a live audience? Once? Hundreds of times? What if it works most times, but fails if certain numbers are chosen? We must check every possibility. We can work out how many with a bit of maths. There are 9 spades and 9 hearts. Any pair could come up, so we have to check every heart with every spade. That is 9 hearts to test for every spade chosen, so 9 x 9 possibilities or 81 tests in total. We must calculate 81 friendship numbers to be sure!

Hospital Tests

The same problem arises with programs. When a computer crashes, you've found a situation the programmers didn't test for. Programs are much more complicated than magic. Think about those in hospitals again. The nurse might have to enter a volume of drug like 28.8 mL and a rate (how quickly the drug should be given) like 1.2 mL per hour. There aren't just 9 possibilities for each number. If numbers up to 999 with one decimal place are allowed there are 10000. That would be 10000 x 10000 or 100 million possibilities. All those combinations can't be checked. It is worse than that. Other things have to be entered too (like units) – more combinations each time. Perhaps the program only goes wrong for certain doses when using certain units.

It's impossible to test all the combinations, so programmers don't. They use logical thinking to devise a set of values (a **test plan**) they hope will catch any problem, and test those. Even so, programmers spend more time testing their programs than actually writing code.

Test plans

What kind of test plan might we come up with for our magic trick? We might test some 'typical' i.e., middle values like 4 and 5 and 3 and 6. We might also then choose some extreme values like 9 and 9 then 1 and 1. From sad experience programmers know that things often go wrong with numbers round the edges. Being more inventive, we might add to our test plan 2 and 2 in case something could go wrong if the values were the same. We would test all those values and perhaps some more too until we ran out of time (the audience are waiting!). Then we would just hope. Programmers do that too: stop testing when the deadline for delivering the program arises and just hope!

There must be a better way!

The Magical Friendship Test: Maths

The Maths of Magic

There is a better way for finding mistakes in programs. When people's lives are at stake it is important that programmers use it. We need some school maths – algebra. You aren't made to learn it at school just for fun! It gives computer scientists a valuable computational thinking tool to help save lives.

> The words algebra and algorithm are both linked to the great 9th century Muslim scholar Al-Khwarizmi. Algebra gets its name from the Arabic word meaning 'reunion of broken parts' from his book about it. Algorithm is a variation of his name from his book of Indian numeric algorithms.

The first step is to say what our magic trick is supposed to do: we need to give its **requirements**. We need to do this using Maths rather than English so that we are precise about what we mean. We call this writing a **formal specification**.

It's fairly simple for our rune. There are two parts: 1) saying what the algorithm is supposed to calculate and 2) saying what the audience is told. For our rune, after we have done all the calculations we want the following to be true:

```
reveal = friends
```

We want the number given by the chosen cards and the number calculated to be identical.

We also want the presentation to be right. We want at the end:

```
SPEAKETH = "Your friendship will last
for ever"
```

It would be sad if our instructions got the calculation right, then told us to say the wrong thing!

But what does it mean?

The next step is to give our language a mathematical meaning. It's a way programming languages and human ones differ. If we give instructions in English, it's easy for another person to get the wrong idea and do the wrong thing. We can't let that happen with programs, so we use maths.

We need to define mathematically (i.e., precisely) what each part of a program means. Numbers are easy – they just stand for the number themselves. What about variables? They stand for the last number that was linked to them (by an assignment). Arithmetic symbols mean do that calculation on the numbers. So if 4 was the last thing assigned to `spade` then (`10 - spade`) stands for the number 6.

An assignment like:

```
a <~ c
```

means set the variable `a` to be the answer to the calculation, `c`. After it happens we know the fact: `a = c`.

The Magical Friendship Test: Maths

Proving Magic Works

Let's prove the trick program works. Our first assignment was

```
& spade <~CHOOSETH{1,2,3,4,5,6,7,8,9}&
```

CHOOSETH picks one of the numbers in its list at random. We can't say which number as it will be different every time. Let's just call it **s**. Given our rule about how assignment works, after the above command we will know that:

spade = s

Actually we know more. We also know the number that CHOOSETH gives is one of the numbers in its list, here 1 to 9. So we can say the following will become true:

(spade = s) AND (s ≥ 1) AND (s ≤ 9)

We are using AND to mean a very precise mathematical operation. The logical statement a AND b is true when both a is true and b is true. It is false if either a or b are false.

In the next step the second friend picks a heart, which we will call **h**. After that assignment we know the following is true.

(spade = s) AND (heart = h) AND

(h ≥ 1) AND (h ≤ 9) AND (s ≥ 1) AND (s ≤ 9)

The Magical Friendship Test: Maths

What effect does the next step of our trick have?

```
& magic <~ 10 - spade &
```

This says to work out what `10 - spade` is and make `magic` equal to that value. So immediately after it has executed we know a new fact that is true:

```
magic = 10 - spade
```

Our description of what we are sure about the world becomes

(`spade` = s) AND (`heart` = h) AND

(`magic` = 10 – `spade`) AND

(h ≥ 1) AND (h ≤ 9) AND (s ≥ 1) AND (s ≤ 9)

That is the state of the world when we do the next step of the trick

```
& total1 <~ heart x 2 &
```

It sets `total1` to be equal to the result of the calculation: `heart` x 2

This gives us a new fact to add:

(`spade` = s) AND (`heart` = h) AND

(`magic` = 10 – `spade`) AND

(`total1` = `heart` x 2) AND

(h ≥ 1) AND (h ≤ 9) AND (s ≥ 1) AND (s ≤ 9)

Though this looks similar to our rune, it was a program giving instructions of things to do in order in a programming language. This is saying what the result of following those instructions is in logic. It is a series of mathematical equations.

The Magical Friendship Test: Maths

Solving the Trick

If we keep going, we get a logical statement of what is true once all the assignments are done:

(`spade` = s) AND (`heart` = h) AND

(`magic` = 10 – `spade`) AND

(`total1` = `heart` x 2) AND

(`total2` = `total1` + 2) AND

(`total3` = `total2` x 5) AND

(`friends` = `total3` – `magic`) AND

(`reveal` = `heart` ^ `spade`) AND

(h ≥ 1) AND (h ≤ 9) AND (s ≥ 1) AND (s ≤ 9)

Now what do mathematicians do with equations? They combine them, getting rid of some of the variables. We can do that one equation at a time. As we know `spade` = s then we can replace `spade` with s in all the other equations so that it doesn't appear any more. Similarly we can replace `heart` with h. Mathematicians write things like a x 2 as 2a so lets do that here too. We get the simpler set of equations:

(`magic` = 10 – s) AND

(`total1` = 2h) AND

(`total2` = `total1` + 2) AND

(`total3` = `total2` x 5) AND

(`friends` = `total3` – `magic`) AND

(`reveal` = h ^ s) AND

(h ≥ 1) AND (h ≤ 9) AND (s ≥ 1) AND (s ≤ 9)

We can do the same, with the next two equations replacing `magic` with 10 – s and `total1` with 2h everywhere they appear, giving:

(`total2` = 2h + 2) AND

(`total3` = `total2` x 5) AND

(`friends` = `total` - (10 - s)) AND

(`reveal` = h ^ s) AND

(h ≥ 1) AND (h ≤ 9) AND (s ≥ 1) AND (s ≤ 9)

Eliminating `total2` and `total3` gives:

`friends` = 5(2h + 2) – (10 – s) AND

`reveal` = h ^ s AND

(h ≥ 1) AND (h ≤ 9) AND (s ≥ 1) AND (s ≤ 9)

Now we can simplify that complicated equation about `friends` as follows:

5(2h + 2) – (10 – s) =

5(2h + 2) – 10 + s =

10h + 10 – 10 + s =

10h + s

So we can replace 5(2h + 2) – (10 – s) with 10h + s and our facts simplify to:

`friends` = 10h + s AND

`reveal` = h ^ s AND

h ≥ 1 AND h ≤ 9 AND s ≥ 1 AND s ≤ 9

The Magical Friendship Test: Maths

Reveal the truth

What number does h ^ s turn in to? When we put two numbers side by side they become a 2-digit number – as long as they are just digits and not bigger numbers. We need our facts about the possible numbers selected: $h \geq 1$, $h \leq 9$, $s \geq 1$ and $s \leq 9$. That is just a mathematical way of saying they are non-zero digits. The trick won't work if larger numbers are chosen!

What do we mean by a two-digit number? Just that one column stands for 10s and the other for units. In the number 54, the 5 isn't just 5. It stands for 50. So saying we have a 2-digit number just means that the digit in the 10s column (h) is multiplied by 10 and added to the units digit (s). We can write this as:

h ^ s = 10h + s.

So, for example, 5^4 (5 placed next to 4) turns into (10 x 5) + 4 = 54. Using that new fact we get:

`friends` = 10h + s AND

`reveal` = 10h + s

Together they give a single, simple fact that is true at this point in the trick:

`friends` = `reveal`

It says that after we have done all the calculations the two numbers, `friends` and `reveal` are identical. It is true whatever digits h and s are!

That is the state of the world before the If statement is carried out. It does the test:

{ `reveal EQUALETH friends` }

but `EQUALETH` just checks if two things are the same. It is asking whether `reveal = friends` is true or not. We just proved it is always true, so the rune will always do the first instruction, never the second. We will always execute

```
& SPEAKETH <~
 "Your friendship will last for ever" &
```

The fact we will know about the world after the IF instruction is followed is that

```
SPEAKETH =
 "Your friendship will last for ever"
```

We have proved the trick always works using logical thinking. We have shown that the friendship number and revealed number will always be the same and so we will never be embarrassed. We will always say:

"Your friendship will last for ever".

The truth about a program

The specification for a medical device program would be similar. Rather than being about what a magician says it would be about what appears on the screen, what lights are lit up and what sounds are made. We can prove a program meets its specification just like we did with the trick.

The Magical Friendship Test: Design

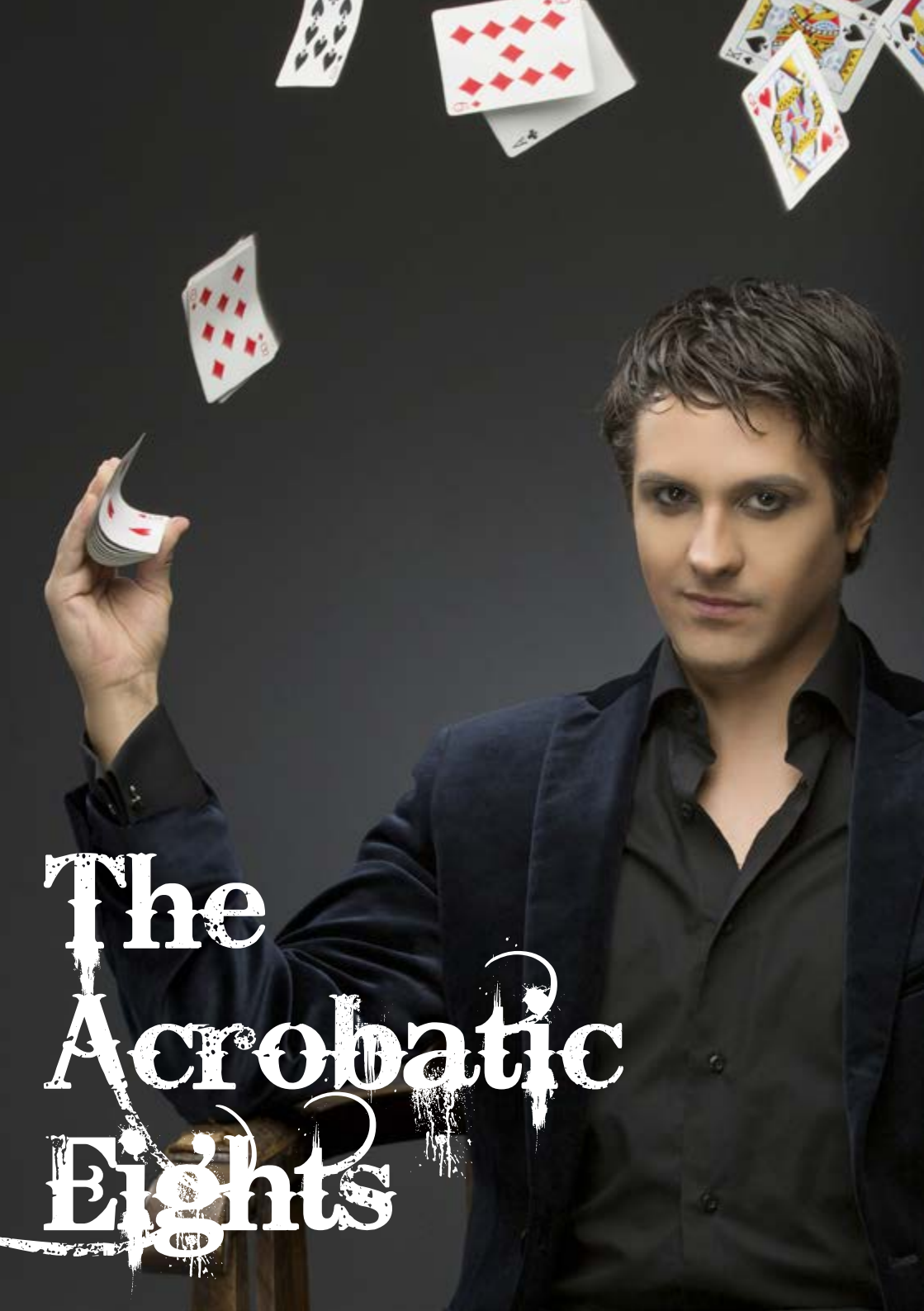
Human Error

We proved the trick always works. This of course assumes the magician and volunteers follow the steps exactly. If the volunteer gets the calculation wrong or writes down a different number the trick won't work whether we have proved it or not. That is entirely possible as the previous trick showed. We should make sure the trick takes account of the possibility.

We want a plan that is better than just saying the friends will fall out. We could, for example, add steps to check the calculation has been done correctly: perhaps have both friends do the calculation and compare answers, only writing the final number down when they agree. Maybe you can come up with a better way. The real point here is to think about what could go wrong and build ways to prevent them happening in to the trick. That way you will recover if they do.

The same applies to programs. If the user of a program does something that the programmer hadn't thought of, then the program may not do the right thing. That is why it is really, really important to program in a way that assumes people will do the wrong thing. You should also evaluate for whether something sensible and safe happens if a user does get it wrong.

For example take medical devices again. Suppose a nurse or doctor makes a mistake doing a calculation and enters a number 10 times too big to be safe, or enters a number that doesn't make sense like one with two decimal points. The program ought to be able to detect that and issue a warning and not go on until they have corrected the mistake. Just as with the trick it would be better still to have a way of preventing the mistake being made in the first place. It is standard practice in hospitals for a second nurse to check any calculation done. Different people, it turns out often make the same mistakes though so actually something more is needed. The programmers need to come up with ways their programs can help.


The
Acrobatic
Eights

The Acrobatic Eights: Magic

Most people think the Aces do all the best tricks. Here the eights show themselves to be the acrobats of the pack by invisibly back flipping tumbling and diving from one pile to another.

The mechanics

This trick uses one simple bit of secret card manipulation, turning the pack over. The routine ensures this easy pack acrobatic is hidden through audience misdirection.

Going backwards to the set up

Find the four eights from a pack of cards, tossing them causally on the table. Explain they are the pack's acrobats. Ask a spectator to shuffle them. While everyone's attention is on those eights, take the face down pack and secretly reverse the bottom half. You will have plenty of time, as all attention will be on the shuffling eights. Don't rush it or look like you are up to something. Be casual.

With the pack in your hand take the eights back, face down. Hold these four cards in the fingertips of the same hand as the pack and spread them with your other hand. Ask your spectator to choose one and place it, the "leader eight", face down on the table. Close up the fan of the three eights and put them on top of the pack. Then, pointing with the index finger of the hand holding the pack, ask a spectator to reveal the value of the leader card in the middle of the table.

The first hidden twist

This is where you do the first hidden twist. As you and audience watch the eight being revealed, place the pack casually aside. As you table the pack turn it over. Now random cards from the reversed bottom of the pack are on top. The three eights are on the bottom.

Immediately move in with both hands to place the now face up leader eight at "just the right position" on the table, pretending to flick fluff from the "performance area". Don't take too long, only enough time to justify having to put the deck down to use both hands.

Practice turning the pack until you can do it quickly, casually and without looking guilty! It helps if you hold the cards lightly in the bend of you clasped fingers, and as you rotate your wrist to place the pack on the table the wrist move also automatically rotates the pack. The audience will be misdirected from it anyway as its over to one side of the table, away from the interesting stuff of turning the eight over.

The Acrobatic Eights: Magic

A second twist please

Pick up the deck and deal the top three cards face down in a line in the middle of the table. These cards, you remind the audience, are the other three eights (except they aren't as you secretly flipped the pack). Keeping the pace fast so no one checks they are eights, push off three more cards from the deck and tilt them up showing their faces, then pop them face down onto the top of a face down pretend 'eight'. Do the same for the remaining two piles. Don't be tidy, just quick.

Now tidy the three piles of cards. Use the index finger of the hand containing the deck to start to square one of the piles. Then 'realise' putting the pack aside will be faster. As you table the pack to one side flip it over again. The audience are too busy watching the activity in the centre of the table to see.

Pick up the pack and, as before, push over three random cards to place on the face up leader eight. Don't show their faces this time: they are the three eights returned from the bottom to the top of the pack by your second hidden twist. Because you use the same actions, thinking back the audience will assume you showed their faces too, but you didn't. They will confuse this similar event with the three earlier occasions where the faces of the pushed off cards were deliberately shown.

All the fun of the fair

The trick is over for you before it has even started for the audience. They think there are three random cards on top of the leader and three piles containing eights in a line. Now have some fun pretending to move the eights from their piles to the leader pile.

Get the first eight to do a "back-flip": take the pile, turn it over and slap it down, spreading the cards… "Blam, the eight is gone". Turn over an eight on the leader pile. It has back-flipped invisibly over the table to land by the leader.

Go for a tumble with the next pile. Shuffle it then flick one end. Turn over the cards to show the eight has vanished, only to tumble invisibly into the leader pile when you turn over the second eight there.

Make the final eight do a high dive. Take the pile and lift it up, pause for effect, then lift it higher. Flick the cards then show the faces, the final eight is gone. Dramatically turn over the last eight on the leader pile to show the high dive landed on target. The trick is over.

The Acrobatic Eights: Psychology

The Psychology of Misdirection

All tricks combine an algorithm with a presentation. The presentation here is based on an understanding of psychology. It uses misdirection. Misdirection is common in magic: it means your audience pay attention to one thing while you secretly do another.

People are easily distracted. If a magician can make you turn away, then obviously you won't see what she does behind your back. Misdirection is more powerful than that though. Magicians do things that are there to be seen (like turning the pack) in plain sight of a whole audience.

What's going on? There is too much information coming via your eyes for your brain to process everything, so it doesn't try. You have a focus of attention. It is the area that your brain is paying attention to. When you are paying attention to one thing you don't actually see other things in your field of view unless something draws your attention to them. Your focus of attention can be quite small. Misdirection makes use of this so people miss things because their attention is elsewhere.

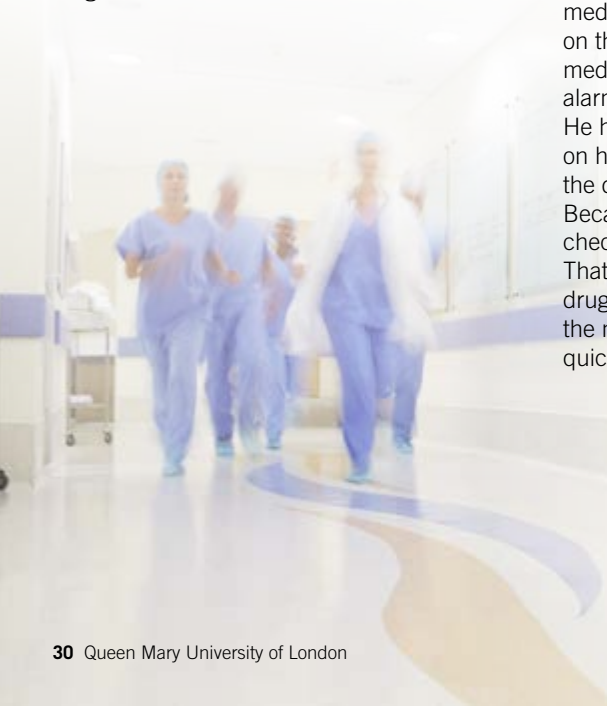
There are lots of ways to be distracted. Trying to do several things at once is difficult and one task can easily distract you from another. Here everyone is given the new task of seeing what card is revealed. The person turning the card has to think about doing it. Pointing is also a strong external signal that directs the attention of the audience. It combines with the natural internal desire to see what the card is. Their attention can't be in two places at once and both internal and external signals are drawing it away from what matters.

Magicians use their understanding of psychology to engineer a system so that a whole audience make the same mistake at once. Everyone looks away from the place that matters at the critical time.

The Acrobatic Eights: Psychology

Misdirection in hospital

Hospitals are busy places. Distraction is a part of the job. Nurses have to do lots of things at once. It's not only checking on and caring for patients themselves. All those machines need to be monitored too. Alarms go off constantly. Sometimes they signal an emergency – a patient having a cardiac arrest, perhaps. Other alarms just mean a patient has rolled over in bed and lent on a tube, causing an alarm to beep in case it is a permanent blockage. Machines can be pretty rude when alarmed – like a baby screaming for its mother. Until they get a nurse's attention they just keep doing it. Alarms, emergencies, patients, pagers, other staff can all misdirect attention just like a magician's misdirection.



Engineering Errors In?

Magicians show it is easy to engineer situations where everyone – hundreds of people – all make the same mistake at the same time. They make the mistake even when they are trying hard not to! Hospitals naturally have the conditions that a magician carefully engineers in to their performance. They are a place where mistakes will be made just as a theatre controlled by a magician is. You can't stop yourself being fooled by a magician, and similarly no nurse or doctor can guarantee never to make a mistake, how ever hard they try. We all only have limited mental resources.

Joe, a busy nurse, sets the last of the machines that will give a baby life-saving medicine. He intends to set the dose as written on the prescription: 123.1 milligrams of the medicine per hour. Just as he is finishing an alarm goes off on a machine of another baby. He hits the start button and rushes to check on her. Unfortunately he doesn't notice that the display showed 1231 not 123.1 mg/hr. Because of the new emergency he does not check on the first baby for several minutes. That baby is given 10 times too much of the drug in that time. Seeing his mistake he stops the machine. Luckily this time, he got back quickly enough.

The Acrobatic Eights: Computational Thinking

Programming Misdirection

So why should computer scientists care? Because well-designed machines will help nurses and doctors avoid mistakes. If programmers don't design for this kind of problem then instead they will cause mistakes to be made. Psychologists do experiments to work out what our limits are. Computer scientists need to use the results when they write programs.

What happened in our example with the nurse Joe? When typing in the numbers his attention was mostly drawn to the keys to find the right digit, not to the screen. He did look but decimal points are small and seeing one is missing is very hard. The position of the start button at the bottom of the machine drew Joe's attention away from the screen not towards it. Worse just as he was checking, the emergency alarm took his attention completely away.

Why was the decimal point missing? Well, on that machine decimal points are only allowed with numbers less that 100 so the decimal point was just silently ignored.

Is this all Joe's fault for "not paying more attention"? If you think that then when you go to a magic show you should see through all the tricks! Whose fault it is should not be the issue, anyway. What matters is working out how things can be redesigned so that accidents waiting to happen don't happen again. Ideally you would have more, less busy, nurses. We can't reprogram people. The programmers can do something to help though.

Design mistakes away

Programs can prevent mistakes and also help a nurse recover when they do make a mistake – before anything bad has happened.

The machine should not silently ignore key presses. It should warn the user and not continue until they have fixed the mistake. The program can work like a magician in reverse, drawing the person's attention to the right place, rather than the wrong place, by changing colour or flashing perhaps.

Why does a decimal point have to be so small? It could be larger.

We could change the way numbers are entered. Rather than use a digit keypad, we could have up and down buttons. Then Joe would just keep his finger on one button to change the number. His attention would be on the screen all the time not his fingers.

The machine could check the dose entered and warn the nurse if dangerous.

Perhaps machines could give fewer false alarms, so they aren't constantly drawing the nurses' attention away from their main tasks. If a tube is temporarily blocked but then clears maybe the program can safely work it out for itself.

A good designer might have lots more ideas for improvement. Of course any changes would need to be tested to make sure they did work. Otherwise they might just introduce new problems.

Between the Two Red Queens

Between the Two Red Queens: Magic

The red queens are detectives. A volunteer chooses a villain card, while you deal out some suspect cards to make the queens' job more difficult. Spelling out your orders to the queens, the villain card is mysteriously trapped between them.

The mechanics

Place the two red queens face up on the table. Have a volunteer secretly remove, and remember, a card from the pack. Deal 12 other random cards on top of it, all face down. Place one red queen face up on the bottom of this pile, and the other face up on the top.

Next, deal a card from the top of the pack for each letter of the word BETWEEN, i.e. deal 7 cards. Drop the remaining cards on top of those dealt out. Following the same procedure, spell-deal THE, then TWO, then RED, and finally QUEENS. Spread the cards and you will find a single face down card sandwiched between the face up queens: the chosen villain card.

All seems sociably sensible

The numbers are hidden by you spelling the magical commands. The trick could be done by counting 7 3 3 3 6, or spelling the 'magic' words JUMPERS BAT BIT EDS WOOLYS. There are also lots of other sequences that end with the villain trapped between the queens. The 'magic' comes from the natural way the numbers are hidden in a phrase that links to the story you are telling: BETWEEN THE TWO RED QUEENS. It makes sense in the context of the trick, and makes sense socially to your audience as a way to command the cards to perform for you. It's also easy to remember.

Between the Two Red Queens: Computational Thinking

A hidden algorithm

The movement of the villain card (let's call it V) is just an algorithm: with the guaranteed effect of leaving it trapped between the two queens, Q.

It is a series of steps that move us from the initial state QVxx xxxx xxxx xxQ to the final state xxxx xQVQ xxxx xxx. We've called all 12 suspect cards x as we don't care what they are. The maths means we always get this outcome if we start in the correct state and carry out the right steps.

This description of the cards is a mathematical model. We have hidden the details that don't matter (what the other cards are) showing the positions of the cards that matter – a form of abstraction.

We can show the trick works using logical reasoning about the model. Go through the spell-deal moves and see how the pattern of card locations in the model changes. Spell-dealing BETWEEN splits the pack into QVxx xxxx and xxxx xx Q. It reverses the latter to Qxxx xxx, then adds QVxx xxxx on the end. It changes the state from QVx xxx xxx xxx xxQ to Qxx xxx xQV xxx xxx. Treat each step as an assignment and prove the trick always works as for the Magical Friendship Test.

Crank up the magic

As it stands this is quite magical. Spell-dealing the words traps the villain. We've performed it and it works. But when you try a new trick with an audience you always see places where people notice weaknesses in the magic. Those places, where there is a suspicion of how it works, could spoil the effect and that wonderful astonishment you want them to have. All magicians need this sort of live feedback. Each time you do a trick, it's an experiment with a live audience, and you try to use that to improve.

In this trick we found the way that the villain card is placed on the bottom of the suspect pack is a problem. It's needed for the trick to work, but it felt suspicious. After all, the story is about finding a hidden villain, and that card is not hidden! It's firmly placed on the bottom.

Hide the villain

We need the audience to believe it is lost in the suspects. How? Well, with the villain at the bottom of the suspect pile you can then use a technique called a false shuffle to shuffle the cards while actually keeping the villain where it needs to be, on the bottom. It works by physics: friction to be precise. If you press your fingertips on the bottom (villain card) of the pile and pull cards from above it with the other hand then the friction of your fingers against the bottom card keeps it in place. You can lift cards from above and move them to the top like a real shuffle while keeping the villain card firmly anchored to the bottom of the pile. Try it and you will see how convincing it is. After a false shuffle to 'hide the villain' the Queens can be placed and the trick completed.

Between the Two Red Queens: Computational Thinking

Don't count your suspects

That improved the trick a lot, but then as we did it more we realised it was a bad idea to count out 12 cards. Sometimes the audience realise that the number matters. We do need 12 suspect cards, but we need it to seem like we are casually adding suspects, as if the exact number doesn't matter. How could we do this? Well, pushing cards over in blocks of three is easy to do and looks quite causal. Four blocks of three gives 12 cards with less suspicion that the trick is about exact numbers. We tried this and realised a bit more helped. If we deal the first three blocks of three, pause and say 'perhaps a few more' before pushing off the final set of three, then it provides a nice socially distracting cue to the apparent casualness of the counting.

Playing with words

The false shuffle removes the idea from the audiences' mind that the trick is mechanical: put card here, spell words, card ends up here. That is exactly how it does work, but we don't want it to be obvious. After all, in magic when you know where the elephant is hidden the wonder goes! Perhaps we can think of other ways to throw them off the trail? We could give them options to choose from. For example, the words we use involve the final RED QUEENS.

We could let them chose the detective cards to use. ODD SEVENS or RED EIGHTS work for the 3 6 sequence. How about ODD THREES, or OLD SWORDS, given the King of Hearts, King of Spades and King of Clubs pictures' have swords. Start the trick by tossing out some random cards for the audience to choose the detectives from: Queens, sevens, threes, eights and Kings.

Experiment – over to you

Experiment. All your variations will work in the sense of trapping the villain, if they are based on the mathematical algorithm, but depending on your presentation and the story you tell the mechanics and the magic effect will be modified. Just keep in mind the best magic is simple and direct. It makes sense to the audience if they can follow the reasons for each move as it happens, then when the impossible happens at the end their bubble of reality bursts and they are astonished and entertained.

Between the Two Red Queens: Computational Thinking

Human-centred algorithmic thinking

The bare mechanics (the algorithm) of all the tricks in this book are easy to see through. It takes good presentation to be magical. Similarly with a program, developing a brilliant algorithm isn't enough. Programs used by people have to have good **interaction design** (i.e., presentation) if they are to be easy and a delight to use. Computer scientists can use their experience to suggest designs, and logical thinking to evaluate them for problems, but ultimately they must be evaluated with people. Computer Scientists draw on methods from the social sciences to do this.

One way to develop a trick is to hide away, only using it in a magic show once perfected in private. That's NOT a good way to do it. Tricks, and even parts of tricks like false shuffles, should be tried out as soon as possible. Before you move to a large audience, try a new trick on friends, one at a time, to see how well it works. Modify it based on what is working magically and what isn't. Then try out the new version on someone else. Just in trying it, your creativity will be sparked and you will think of new tweaks. When it's working well, try it with a bigger audience and then bigger still. Evaluation with people and development go hand in hand. None of the tricks in this book rose out of a dark cellar. They all went through this human-centred process. The same applies to programs.

Between the Two Red Queens: Social Sciences

Useful User Testing

Early in the development process computer scientists try out their designs on real people, checking the ideas. Called **user testing**, it is done on prototypes long before the full program is completed. One way, called **post-it note prototyping** is used on the earliest design ideas. You sketch a picture of the design on paper and use sticky notes for the screen. A volunteer – the 'user' – does a task (like enter a drug dose for a medical machine). As they pretend to press buttons doing it, you swap sticky notes to show what happens on the screen. You watch and listen as the user says what they are thinking and what is confusing. You fix the problems then do more user testing.

A next step is to create a higher-fidelity prototype. You could do this in Powerpoint using 'action buttons' so when you touch a button on a picture of a device prototype it moves to a new slide showing what pressing the button does. This is a bit more realistic. Later still early versions of the real program are user tested until eventually a near-release version is 'beta-tested' on lots of real users.

Problems with the design are then found early when they are cheap and easy to fix. Good algorithmic thinking gives good algorithms, but it takes human-centred algorithmic thinking to create great programs.

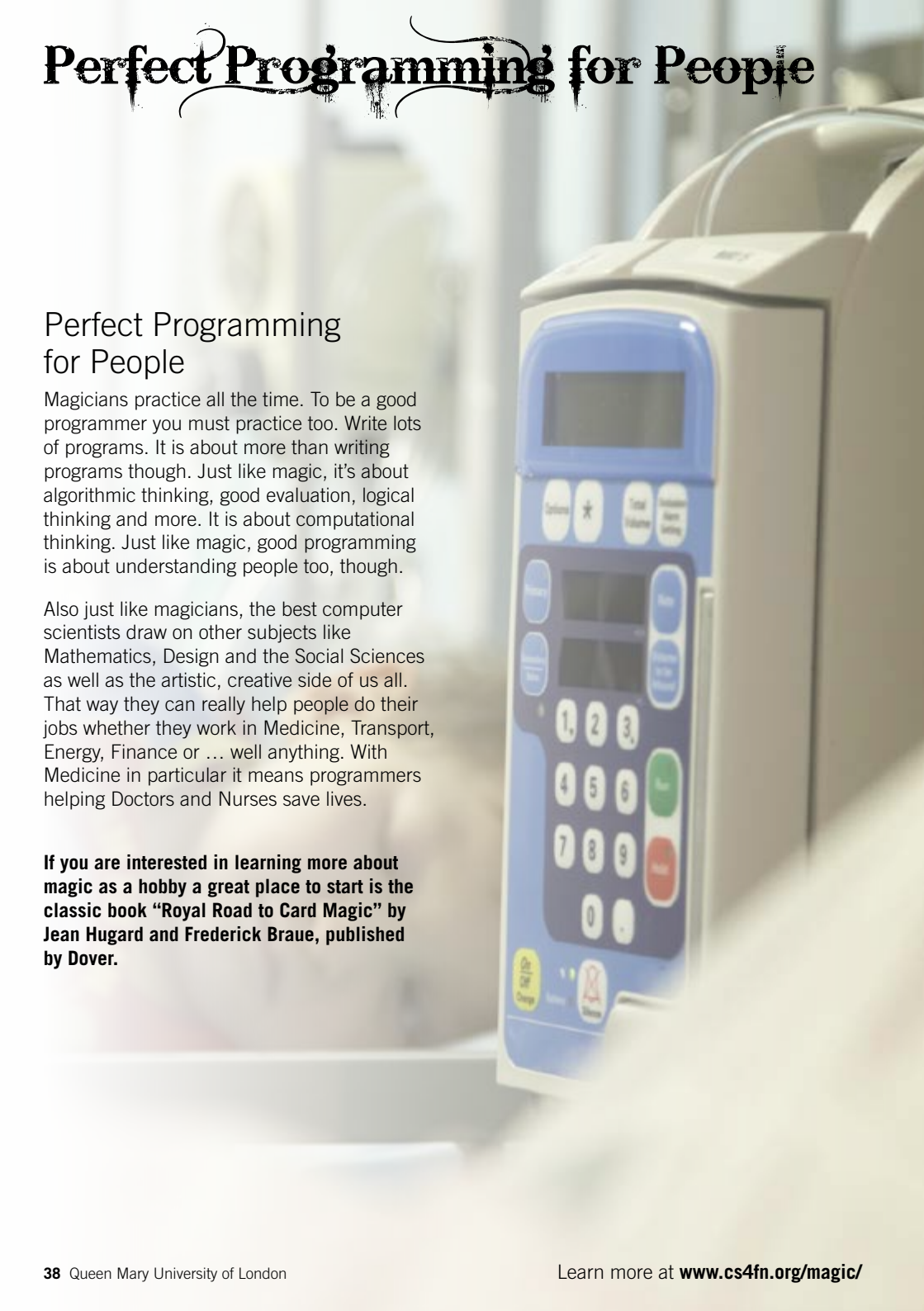
Essential Ethnography

If you want to be a great magician you need to practice a lot. You also need to watch magic – see the same tricks being done over and over. Don't just watch the magician, watch the audience too. See what they do and their expressions as the trick progresses. See what works and what doesn't. Watch people generally too. When do they (and you) make mistakes? What distracts and what doesn't? We noticed a crowd of people were distracted when we handed one a card while chatting. From that observation, (and lots of trial and error) a new trick emerged.

Ethnography is something social scientists do: they observe people getting on with their normal lives to understand them better. It's a powerful tool for computer scientists too. Before you develop a program, get out amongst the intended users (eg in hospitals) and find out what their lives are like, what problems they face and how they do things. Immerse yourself by doing their jobs with them, so you deeply understand. That way you will be in a position to design something that they will use and that will be really useful. Otherwise you may end up creating a 'wonderful' program that no one ever uses because it just doesn't fit the way they do things. Even better you may notice a problem to solve, seeing something that nurses are struggling to do say, that your competitors haven't even realised is a problem. That will give you a marketing edge.

Ethnography can be used to develop a set of personas – very detailed descriptions of fictional people with names, jobs, feelings, lives…doctors, nurses, patients, hospital porters, cleaners,... They are used by the design team to help them remember the real people they are designing for. When deciding if a feature is needed they can ask "Would it actually help Sue or Joe".

Perfect Programming for People



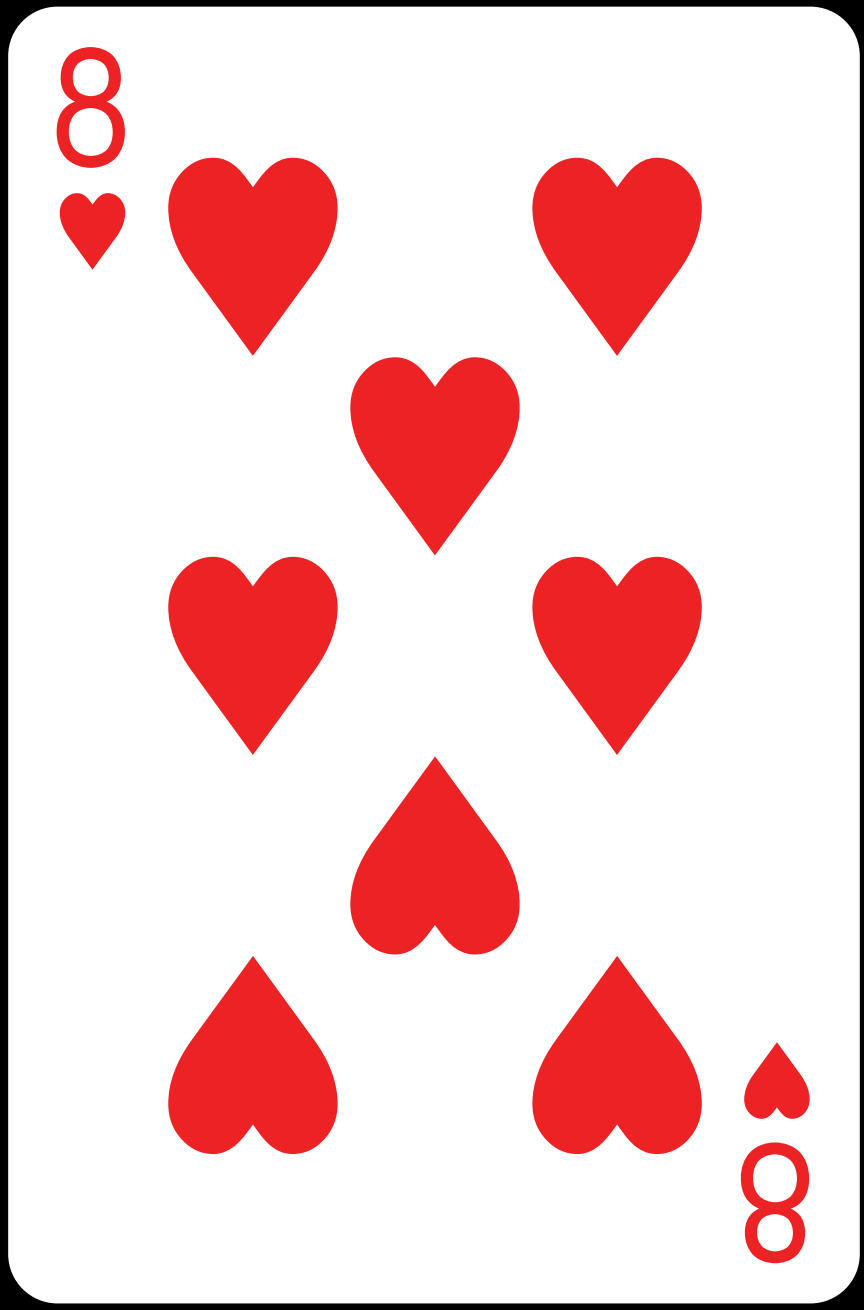
Perfect Programming for People

Magicians practice all the time. To be a good programmer you must practice too. Write lots of programs. It is about more than writing programs though. Just like magic, it's about algorithmic thinking, good evaluation, logical thinking and more. It is about computational thinking. Just like magic, good programming is about understanding people too, though.

Also just like magicians, the best computer scientists draw on other subjects like Mathematics, Design and the Social Sciences as well as the artistic, creative side of us all. That way they can really help people do their jobs whether they work in Medicine, Transport, Energy, Finance or … well anything. With Medicine in particular it means programmers helping Doctors and Nurses save lives.

If you are interested in learning more about magic as a hobby a great place to start is the classic book "Royal Road to Card Magic" by Jean Hugard and Frederick Braue, published by Dover.

Learn more at **www.cs4fn.org/magic/**



This is your chosen card

The Magic of Computer Science
www.cs4fn.org/magic/

Queen Mary
University of London

This cs4fn booklet (March 2015) is part of a project between Queen Mary University of London and Hertford College Oxford. It was funded by the Department for Education with additional support provided by Google and the EPSRC funded CHI+MED project involving UCL, Swansea and City Universities. It was distributed in London with support from the Greater London Assembly.

For resources on the fun side of computing, visit **www.cs4fn.org**. For courses and resources for teachers, including classroom activities linked to this booklet, visit **www.teachinglondoncomputing.org** a joint project between Queen Mary University of London and King's College London funded by the Greater London Assembly.

Department for Education

SUPPORTED BY
MAYOR OF LONDON

COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE

chi+med

EPSRC
Engineering and Physical Sciences Research Council