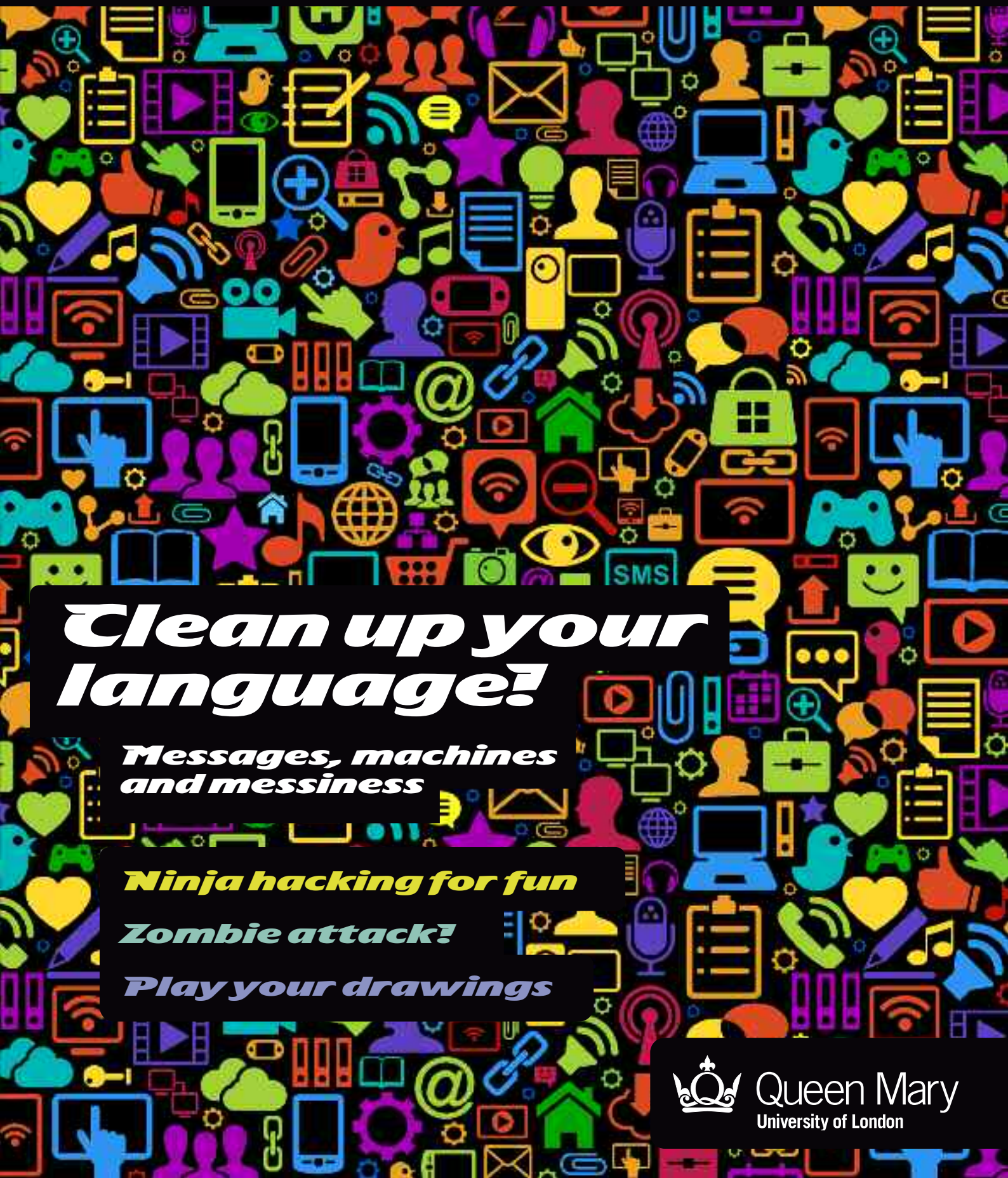
ISSN 1754-3657 (Print)
ISSN 1754-3665 (Online)

CS4FN

Computer Science for Fun

Issue 16



Clean up your language!

Messages, machines and messiness

Ninja hacking for fun

Zombie attack!

Play your drawings

Queen Mary University of London

Language in the machine

It may seem odd to the uninitiated but language – that most human of inventions – is also a core topic of computer science. Computers are just machines that can fctions – programs. Those instructions have to be written in special languages (programming languages) that are precise and unambiguous. New languages are being invented all the time each designed to simplify different bits of computing. Some make it easier to write programs while others are used to check that computer systems behave the way they should.

Language isn't just the thing that makes computers tick. A lot of exciting computer science helps them work with human languages. The earliest computers were actually developed to do exactly that – in World War II to allow the Allies to unravel the complex German secret codes. Now with Internet technology, computer science is inextricably linked with the way we communicate. It helps us send messages to each other and can also to hide messages from snoopers. Computers help us translate from one language to another and may even help to save threatened languages. Language is the essence of computer science.

The heart of an Arabic programming language

So far all computer languages have been written in English, but that doesn't need to be the case. Computers don't care. Computer scientist Ramsey Nasser has developed the first programming language that uses Arabic script. His computer language is called بلق. In English, it's pronounced "Qalb", after the Arabic word for heart. As long as a computer understands what to do with the instructions it's given, they can be in any form, from numbers to letters to images.

I learn Greek, you can too

There's an app for that! Learning a new language is never easy, but Anthony Ladis, a student at Queen Mary, University of London, is helping primary school pupils to get a grip on Greek. His final year project is a free app, iLearnGreek. It tests understanding of simple Greek words, like clothes, colours and the seasons, in a series of fun games. It also lets kids write Greek letters to win virtual trophies. Real pupils and teachers who tested the app felt it really helped make learning fun. Download it for free from the App Store!

Find more free apps from Queen Mary at **www.qappsonline.com**

Ninja hacking for fun

Computer hackers are the bad guys, aren't they? They cause mayhem: shutting down websites, releasing classified information, stealing credit card numbers, spreading viruses. They can cause lots of harm, even when they don't mean to. Not all hackers are bad though. Some, called white-hat hackers, are ethical hackers, and the best are paid by companies to test their security by actively trying to break in with permission – it's called penetration testing. It's not just business though, it's also now a game.

Perhaps the most famous white-hat hacker is Kevin Mitnick. He started out as a bad guy – the most wanted computer criminal in the US. Eventually the FBI caught him, and after spending five years in prison he reformed and became a white-hat hacker who now runs his own computer security company. The way he hacked systems had nothing to do with computer skills and everything to do with language skills. He did what's called social engineering. A social engineer uses their skills of persuasion to con people into telling them confidential information or maybe even actually doing things for them like downloading a program that contains spyware code. Professional white-hat hackers have to have all round skills: network, hardware and software skills, as well as social engineering ones. They need to understand a wide range of potential threats if they are to properly test a company's security and help them fix all the vulnerabilities.

Breaking the law and ending up in jail, like Kevin Mitnik, isn't a great way to learn the skills for your long-term career though. A more normal way to become an expert is to go to university and take classes. Wouldn't playing games be a much more fun way to learn than sitting in lectures, though? That was what Tamara Denning, Tadayoshi Kohno, and Adam Shostack, computer security experts from the University of Washington, wondered. As a result, they teamed up with Steve Jackson Games and came up with Control-Alt-Hack™. It's based on the cult tabletop card game, Ninja Burger®.

Rather than being part of a Ninja Burger® Delivery team, in Control-Alt-Hack™ you are an ethical white-hat hacker working for an elite security company. You have to complete missions using your Ninja hacking skills: from shutting down the US telephone network to turning a robotic vacuum cleaner into a pet. The game is lots of fun, but the idea is that by playing it you'll learn lots about the kinds of threats that security experts have to protect against.

So if you like gaming why not learn something useful at the same time as having fun? Who knows, it might even lead one day to a career as a security expert.

Try their game by visiting **www.controlalthack.com**

The Chinese room: zombie attack!

Iain M Banks's science fiction novels about 'The Culture' imagine a universe inhabited (and largely run) by 'Minds'. These are incredibly intelligent machines – mainly spaceships – that are also independently thinking conscious beings with their own personalities. From the replicants in Blade Runner and robots in Star Wars to Iain M Banks's Minds, science fiction is full of intelligent machines. Could we ever really create a machine with a mind: not just a computer that computes, one that really thinks? Philosophers have been arguing about it for centuries. Things came to a head when philosopher John Searle came up with a thought experiment called the 'Chinese room'. He claims it gives a cast iron argument that programmed 'Minds' can never exist. Are the computer scientists who are trying to build real artificial intelligences wasting their time? Could zombies lurch to the rescue?

The Shaolin warrior monk

Imagine that the galaxy is populated by an advanced civilisation that has solved the problem of creating artificial intelligence programs. Wanting to observe us more closely they build a replicant that looks, dresses and moves just like a Shaolin warrior monk (it has to protect itself and the aliens watch too much TV!) They create a program for it that encodes the rules of Chinese. The machine is dispatched to Earth. Claiming to have taken a vow of silence, it does not speak (the aliens weren't hot on accents). It reads Chinese characters written by the earthlings, then follows the instructions in its Chinese program that tell it the Chinese characters to write in response. It duly has written conversations with all the earthlings it meets as it wanders the planet, leaving them all in no doubt that they have been conversing with a real human Chinese speaker.

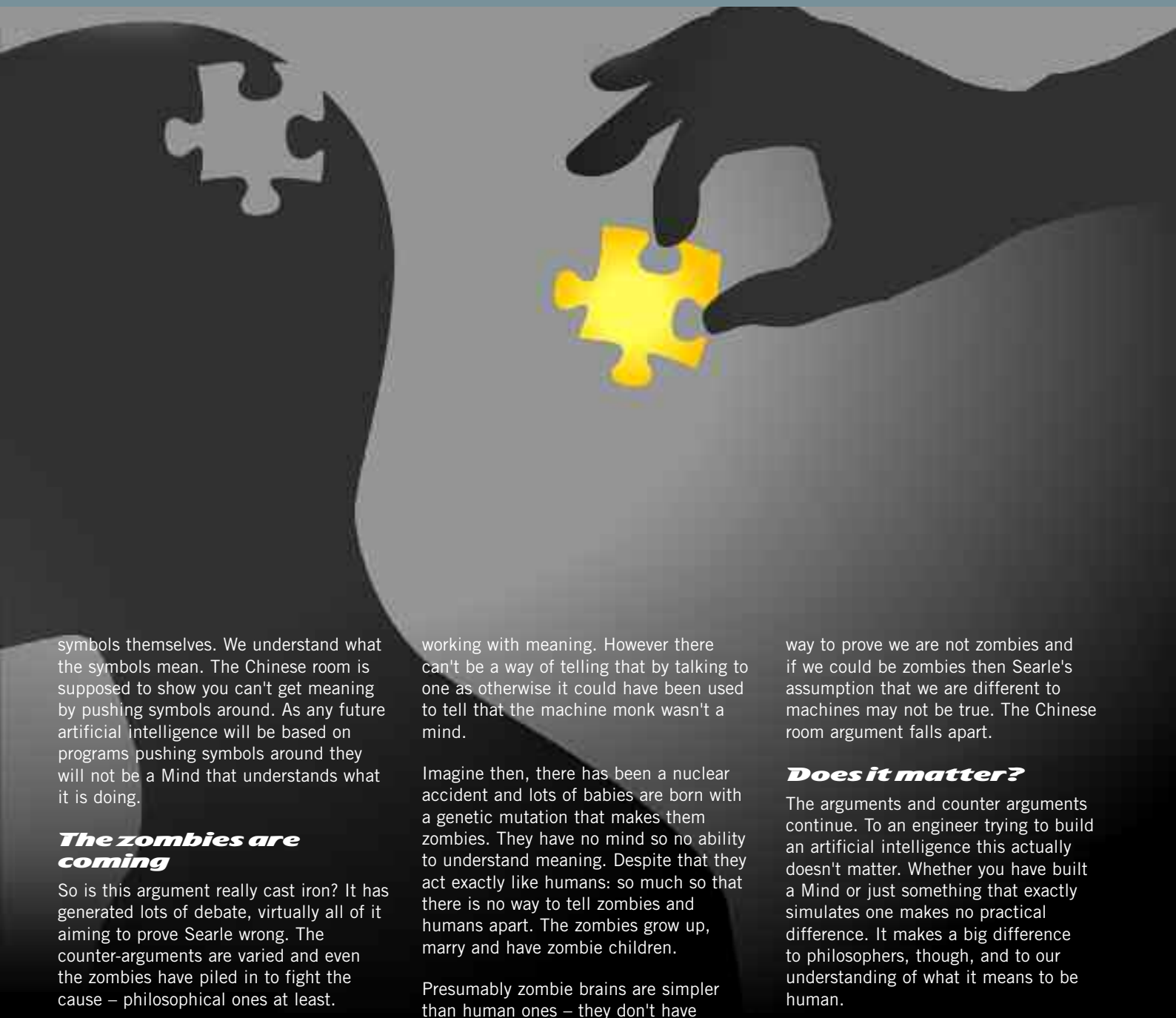
The question is, is that machine monk really a Mind? Does it really understand Chinese or is it just simulating that ability?

The Chinese room

Searle answers this by imagining a room in which a human sits. She speaks no Chinese but instead has a book of rules – the aliens' computer program written out in English. People pass in Chinese symbols through a slot. She looks them up in the book and it tells her the Chinese symbols to pass back out. As she doesn't understand Chinese she has no idea what the symbols coming in or going out mean. She is just uncomprehendingly following the book. Yet to the outside world she seems to be just as much a native speaker as that machine monk. She is simulating the ability to understand Chinese. As she's using the same program as the monk, doing exactly what it would do, it follows that the machine monk is also just simulating intelligence. Therefore programs cannot understand. They cannot have a mind.

Is that machine monk a Mind?

Searle's argument is built on some assumptions. Programs are 'syntactic devices': that just means they move symbols around, swapping them for others. They do it without giving those symbols any meaning. A human mind on the other hand works with 'semantics' – the meanings of symbols not just the



symbols themselves. We understand what the symbols mean. The Chinese room is supposed to show you can't get meaning by pushing symbols around. As any future artificial intelligence will be based on programs pushing symbols around they will not be a Mind that understands what it is doing.

The zombies are coming

So is this argument really cast iron? It has generated lots of debate, virtually all of it aiming to prove Searle wrong. The counter-arguments are varied and even the zombies have piled in to fight the cause – philosophical ones at least.

Could zombies lurch to the rescue?

What is a philosophical zombie? It's just a human with no consciousness, no mind. One way to attack Searle's argument is to attack the assumptions. That's what the zombies are there to do. If the assumptions aren't actually true then the argument falls apart. According to Searle human brains do something more than push symbols about – they have a way of working with meaning. However there can't be a way of telling that by talking to one as otherwise it could have been used to tell that the machine monk wasn't a mind.

Imagine then, there has been a nuclear accident and lots of babies are born with a genetic mutation that makes them zombies. They have no mind so no ability to understand meaning. Despite that they act exactly like humans: so much so that there is no way to tell zombies and humans apart. The zombies grow up, marry and have zombie children.

Presumably zombie brains are simpler than human ones – they don't have whatever complication it is that introduces minds. Being simpler they have a fitness advantage that will allow them to out-compete humans. They won't need to roam the streets killing humans to take over the world. If they wait long enough and keep having children, natural selection will do it for them.

The zombies are here

The point is it could have already happened. We could all be zombies but just don't know it. We think we are conscious but that could just be an illusion – another simulation. We have no way to prove we are not zombies and if we could be zombies then Searle's assumption that we are different to machines may not be true. The Chinese room argument falls apart.

Does it matter?

The arguments and counter arguments continue. To an engineer trying to build an artificial intelligence this actually doesn't matter. Whether you have built a Mind or just something that exactly simulates one makes no practical difference. It makes a big difference to philosophers, though, and to our understanding of what it means to be human.

Let's leave the last word to Alan Turing. He pointed out 30 years before the Chinese room was invented that it's generally considered polite to assume that other humans are Minds like us (not zombies). If we do end up with machine intelligences so good we can't tell they aren't human, it would be polite to extend the assumption to them too. That would surely be the only humane thing to do.

Play your drawings with TuneTrace

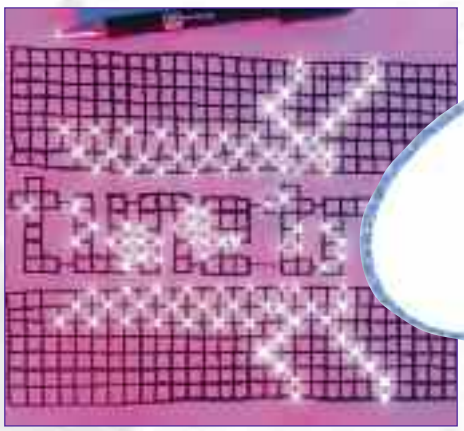
In times gone by, pictograms formed much of human language. The ancient Egyptians used hieroglyphs, pictures representing words and ideas. As the centuries passed, humanity started to make language more abstract. Rather than a picture of a cat, they came up with agreed ways that certain symbols could represent the same thing and that's how CAT came to be. In the dawn of computers, humans used symbols to tell the machines what to do. "LET X= 1.0" told the electronics to set up a place in its memory called X and fill that slot with the number 1.0. The computer would then dutifully comply; X was 1.0 until it was told otherwise.

The idea of using a special language to control what a computer does isn't new; musicians have been using something similar to control orchestras. The musical score, all those funny looking symbols on those long lines, told the players what notes to strike, when and for how long. From simple graphical language music was born and could be transferred so others could play and enjoy.

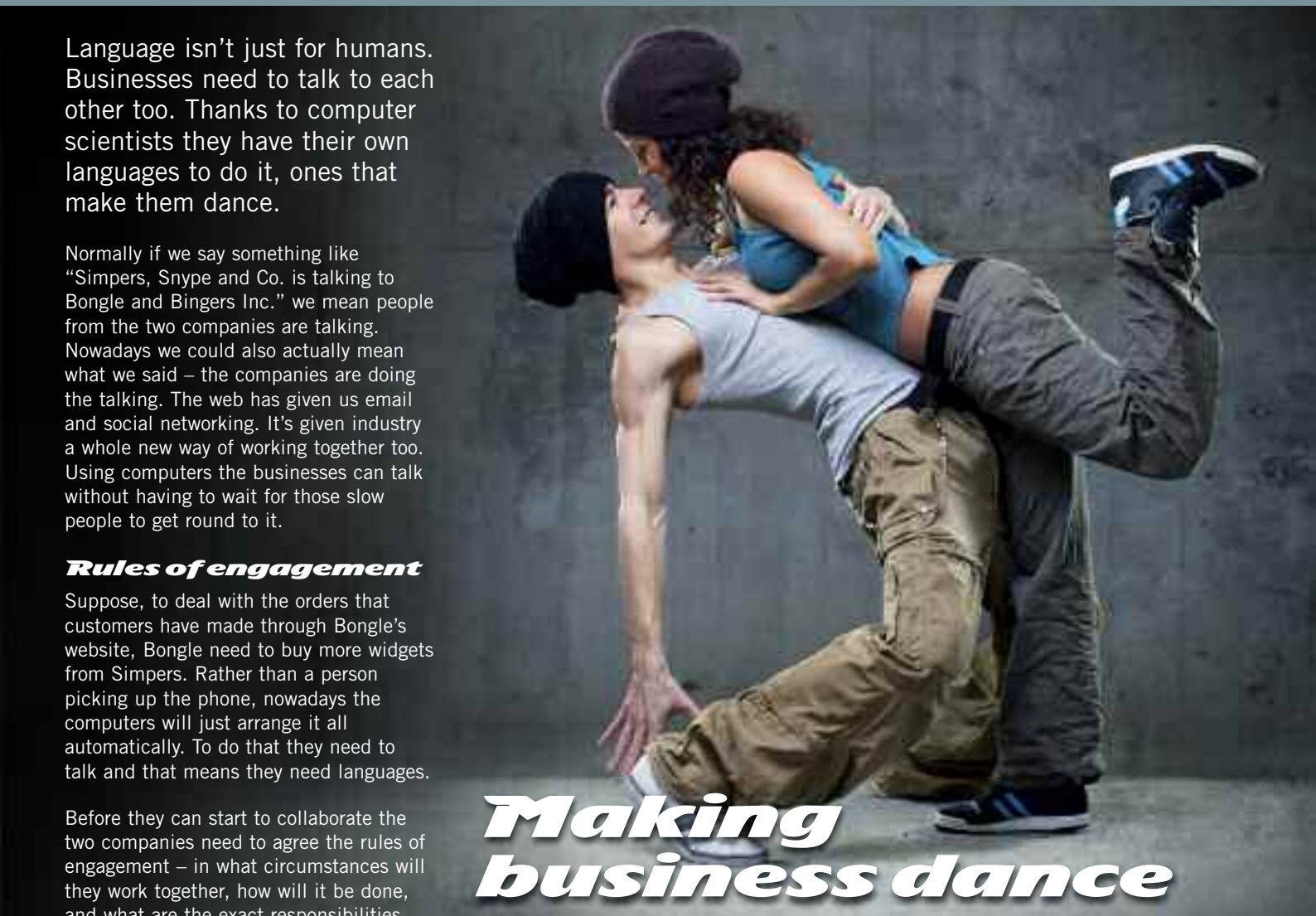
Tunetrace turns your doodle into music

But what if you had the chance to make up your own computer language? You could tell a machine what to do, like how to play a tune. Enter TuneTrace, a smartphone app that lets you explore this in a fun and easy way. TuneTrace takes a line drawing, analyses it and turns your doodle into music. This free app gives you a chance to explore your own musical creativity but also to come up with a way to express your creativity by simply making a drawing. You can share your drawing with others, who can add their own bits and make even more music.

So what will you draw and how will it sound? Download it from the App Store for free and try it!



Find more free apps from Queen Mary at **www.qappsonline.com**



Making business dance

Language isn't just for humans. Businesses need to talk to each other too. Thanks to computer scientists they have their own languages to do it, ones that make them dance.

Normally if we say something like "Simpers, Snype and Co. is talking to Bongle and Bingers Inc." we mean people from the two companies are talking. Nowadays we could also actually mean what we said – the companies are doing the talking. The web has given us email and social networking. It's given industry a whole new way of working together too. Using computers the businesses can talk without having to wait for those slow people to get round to it.

Rules of engagement

Suppose, to deal with the orders that customers have made through Bongle's website, Bongle need to buy more widgets from Simpers. Rather than a person picking up the phone, nowadays the computers will just arrange it all automatically. To do that they need to talk and that means they need languages.

Before they can start to collaborate the two companies need to agree the rules of engagement – in what circumstances will they work together, how will it be done, and what are the exact responsibilities that each is taking on? In the past the rules have tended to be written in English so people can follow them. The trouble with English is it's ambiguous. That's why there is so much work for lawyers! And if one company thought they'd agreed to supply widgets in one size but the other believes the agreement was for whatever size is needed, there's going to be problems.

Dancing the dance

Enter 'Web Services Choreography'. A Choreography in this computing sense doesn't mean managers have to learn to do the tango with each other. It's just a set of agreed rules that tie down exactly what each company will do and when – just like a dance choreography does for dancers. Instead of fixing steps it fixes things like what messages will be sent, by who and when, each time the companies involved collaborate. What message does Simpers' computer need to send to make that order? What does Bongle's computer have to do to confirm it?

It doesn't mean managers have to learn to do the tango

To make it work with computers these rules have to be specified in a precise language. One such language is Scribble, invented by Kohei Honda of Queen Mary, University of London. It is inspired by yet another language – a mathematical language called the pi calculus. Because of its basis in maths, once the rules are set down, mathematical tools can be used to check important things about them. For example, if they follow the rules, is it possible for both computers to end up sitting waiting forever for the other one to do something? Computers can be very patient if that is what their rules tell them to do! If they did it could be disastrous for the companies concerned.

Are we doing it right?

The beauty of choreography is that once the rules of engagement are agreed, and each company is happy with the Scribble version, then they can each get on and implement it on their own computers. Each doesn't have to worry about how the other company is making their own computers do it. The two companies can even write programs to do their bit in completely different programming languages. They can be confident, despite this, that their two computers will be following the agreed rules of engagement and so will be able to work together. This can even be checked mathematically against the original Scribble rules too.

Business is sometimes treated as war, but everyone can benefit if companies work together rather than against each other. Then, at least for computer scientists, it's better to think of it as a dance.

A (com)pressing problem

by Paul Curzon

I have a toddler. I knew I'd signed up for some horrible stuff when we had him: Pooey nappies? Yes. Sleepness nights? Tantrums? Of course. But singing! I never expected so much singing! We sing at his playgroup, we sing watching CBeebies, we even sing at his swimming club.
If you don't like singing it's all rather boringly repetitive. Whether it's Ten Green Bottles, Five Little Speckled Frogs, Row, Row, Row Your Boat or The Wheels on the Bus, you end up singing the same things over and over...and over and over again. If only we could cut some of the repetition! That sounds like a job for some computer science.

A key computer science breakthrough that we all rely on is being able to shrink repetitive things down. Whether it's streaming music or movies, sharing photos or just words, if we are to send large amounts of data across the Internet, we need to do it as efficiently as possible. How? The clue is in those children's songs. Like the songs, most data is highly repetitive and we can make use of that. Before sending a file we can look for repetition and shrink it. It's called data compression. By sending less we can do it quickly.

One of the simplest ways to compress data is to look for sequences of repeated things and just replace them with one copy together with the number of times it appears. This is obviously useful for compressing images. At its simplest, a digital image is just a long series of symbols indicating the colours of each pixel. Since colours tend to stick together in images, the same symbols are likely to appear together a lot. Take that beach photo I took on holiday. It has a bright blue sky with only a wisp of cloud. Suppose blue is represented by B and white by W then the top line of pixels might be something like

BBBBBBBBBBBBBBBBWWWWBBBBBBB

Rather than storing or sending all those characters, we could instead just send:

16B4W8B

It says there is a run of 16 Bs followed by 4 Ws and then 8 Bs. The original 28 characters have been compressed to only 7 without losing any information at all. This kind of compression is called 'run-length encoding'.

Wonderful. So let's look what happens when we apply it to a song.

**"The wheels on the bus go round and round,
round and round,
round and round.
The wheels on the bus go round and round,
all day long."**

Let's look at one very repetitive line for starters:

"round and round"

Applying run-length encoding we get the new version:

1r1o1u1n1d1 1a1n1d1 1r1o1u1n1d

Oh – there's a problem! Even though it is very repetitive, there aren't actually any runs of repeated letters at all. Run-length encoding makes it twice as big not smaller!

The problem is that it is words that are repeated not adjacent characters. We could come up with a way to compress based on repeated words, but let's not give up on run-length encoding yet. Maybe we can improve things.

It turns out we can using a method dreamed up by David Wheeler and Mike Burrows. The way it works is, surprisingly, to first mix all the letters up and then compress the result. The idea is to get as many common letters together before we compress so that run-length encoding can work as well as possible. We have to be cunning in the way we do it though as we need to be able to get the original message back.

The way we do it is to rotate the phrase by one letter - that is moving a letter from the front to the end. "round_and_round" becomes "ound_and_roundr" (where we've used underscores to make the spaces visible). We do that over and over, one rotation at a time, until we get back to the original phrase. For round and round that gives us 15 new phrases:

**round_and_round
ound_and_roundr
und_and_roundro
nd_and_roundrou
d_and_roundroun
_and_roundround
and_roundround_
nd_roundround_a
d_roundround_an
_roundround_and
roundround_and_
oundround_and_r
undround_and_ro
ndround_and_rou
dround_and_roun**

Next we put these into alphabetical order (taking spaces as before 'a' in the alphabet).

**_and_roundround
_roundround_and
and_roundround_
d_and_roundroun
d_roundround_an
dround_and_roun
nd_and_roundrou
nd_roundround_a
ndround_and_rou
ound_and_roundr
oundround_and_r
round_and_round
roundround_and_
und_and_roundro
undround_and_ro**

Finally, we take the last letter of each of these to give the new phrase.

dd_nnnuaurrd_oo

That is the message we compress using run-length encoding. We end up with

2d1_3n1u1a1u2r1d1_2o

It's better than before. Still longer than the original, but let's not worry about that for a moment. Let's worry about whether we can get our original message back given all this shuffling! The answer is yes, as long as you know what the last letter of the original message was. An easy way to do that is just to add a special character at the end of the message, then just rotate, sort and compress it with the rest. You can then reverse each step (maybe you can work out how!)

Why does all this shuffling help, though? The key is that it groups letters that are part of the same repeated pairs together. By sorting the rotated phrases we are grouping those versions of the phrase that start with the same letter. Now think about the last letters. Because we've only done rotations, they are the letters that in the original came just before the first ones. Take the following phrase starting with 'd'.

d_and_roundroun

It ends in an 'n' because the 'd' was originally in the word 'round', where an 'n' came before the 'd'.

If we look at the other phrases that start with 'd', we see there are actually three that end with 'n'.

d_and_roundroun
d_roundround_an
dround_and_roun

That's because in this phrase, the letters 'n' and 'd' appear repeatedly together: twice in the word 'round' and once in 'and'. So because there are three n-d pairs in the original phrase, we end up with a run of three 'n's to compress when we take the last letters.

The same happens with every common pair of letters. The result is that run-length encoding now has lots of runs to work on where before there were none.

We still ended up with something longer than the original though. That's just because the phrase was so short. With longer phrases it gets much better. Try it for yourself with the longer fragment: "round and round, round and round." It is compressed to only 22 characters even though it is more than twice as long as the shorter one we just looked at. The whole verse compresses from 131 to 99 characters. We are getting a big improvement.

With a little bit of computer science we have shortened a long, boringly repetitive song. Much better. So why isn't my toddler impressed when I sing it? On the other hand, streaming of movies or music can only be done quickly enough to play in real time because the data has been compressed first. It's therefore the same kind of clever compression that allows him to watch streamed versions of Thomas the Tank Engine. So while he may not want to sing a compressed song, he ought to sing the praises of compression algorithms.

Hiding in Skype

Computer science isn't just about using language, sometimes it's about losing it. Sometimes people want to send messages so secret that no one even knows the messages exist. A great place to lose language is inside a conversation.

Cryptography is the science of making messages unreadable. Spymasters have used secret codes for a thousand years or more. Now cryptography is a part of everyday life. It's used by the banks every time you use a cashpoint and by online shops when you buy something over the Internet. It's used by businesses that don't want their industrial secrets revealed and by celebrities who want to be sure that tabloid hackers can't read their texts.

Who called who?

Cryptography stops messages being read, but sometimes just knowing that people are having a conversation can reveal too much. Knowing a football star is exchanging hundreds of texts with his teammate's girlfriend suggests something is going on, for example. Similarly, the American CIA chief David Petraeus, whose downfall made international news, might have kept his secret and his job if the emails from his lover had been hidden. David Bowie kept his 2013 comeback single Where Are We Now? a surprise until the moment it was released. The dramatic surprise helped make the single a hit. But the secret could have been spoiled months before if music journalists had noticed Bowie having more conversations with his record label.

Sending messages gets hairy

That's where steganography comes in – the science of hiding messages so no one even knows they exist. Invisible ink is one form of steganography. It was used, for example, by the French resistance in World War II. Steganography has taken more bizarre forms over the years though – an Ancient Greek slave had a message tattooed on his shaven head warning of Persian invasion plans. Once his hair had grown back he delivered it unnoticed.

Digital communication opens up new ways to hide messages. Computers store information using a code of 0s and 1s. Each 1 or 0 is called a bit. Steganography is then about finding places to hide those bits. A team of Polish researchers led by Wojciech Mazurczyk have now found a way to hide them in a Skype conversation.

When you use Skype to make a phone call, the program converts the sounds you make to a long series of bits. They are sent over the Internet and converted back to sound at the other end. At the same time, bits stream back from the person you are talking to, containing the sound of their voice. Data transmitted over the Internet isn't sent all in one go, though. It's broken into packets: a bit like taking your conversation and tweeting it one line at a time.

Commando tactics

Why? Imagine you run a crack team of commandos who have to reach a target in enemy territory to blow it up – a stately home where all the enemy's generals are having a party. If all the commandos travel together in one army truck and something goes wrong along the way, probably no one will make it. The mission would be a disaster. If, on the other hand, the commandos each travel separately and meet once they arrive, the mission is much more likely to be successful. If a few are killed on the way the rest can still complete the mission.

Who'd have thought the sound of silence would be so useful

The same applies to a Skype call. Each packet contains a little bit of the full conversation and each makes its own way to the destination across the Internet. On arriving there, they reform into the full message. To allow this to happen, each packet includes some extra data that says, for example, what conversation it is part of, how big it is and also where it fits in the sequence. If some don't make it then the rest of the conversation can still be put back together without them. As long as not too much is missing, no one will notice.

The sound of silence

Skype does something special with its packets. The size of the packets changes depending on how much data needs to be transmitted. When someone is talking, each packet they send carries a lot of information. When that person is listening Skype sends shorter packets from their end, because they are only transmitting silence. The Polish team realised they could exploit this for steganography. Their program, SkyDe, intercepts Skype packets looking for short ones. When SkyDe finds those short packets, they are replaced with packets holding the data from the covert message. At the destination another copy of SkyDe intercepts them, extracts the hidden message and passes it on to the intended recipient. As far as Skype is concerned some packets just never arrive.

A good hiding

There are several properties that matter for a good steganographic technique. One is its bandwidth: how much data can be sent using the method. Because Skype calls contain a lot of silence SkyDe has a high bandwidth: there are lots of opportunities to hide messages. A second important property is obviously undetectability. The Polish team's experiments have shown that SkyDe messages are very hard to detect. As only packets that contain silence are used and so lost, the people having the conversation won't notice and the Skype receiver itself can't easily tell because what is happening is no different to a typical unreliable network. Packets go missing all the time. Because both the Skype data and the hidden messages are encrypted, someone observing the packets travelling over the network won't see a difference – they are all just random patterns of bits. Skype calls are now common so there are also lots of natural opportunities for sending messages this way – no one is going to get suspicious that lots of calls are suddenly being made.

All in all, SkyDe provides an elegant new form of steganography. Invisible ink is so last century (and tattooing messages on your head is very last millennium). Now the sound of silence is all you need to have a hidden conversation.

Umm, ahh... understanding

Video calls have finally made it. No longer a futuristic dream, anyone can now use Skype. Why bother with face-to-face meetings any more? Just stay in bed and teleconference in to work! Why can't life be that simple for once? It turns out that face-to-face is still better, at least part of the time – video streams are still missing something important.

Computer scientists once believed that if they designed better ways for humans to communicate with computers (good 'human-computer interaction') then they would make it easier for people to communicate with each other too (good 'human-human interaction'). But you don't necessarily make it easy for two people to talk just by making it easy for them both to talk to a computer. You need to understand what makes human conversation work and design for that.

Terrible texting

Text messaging is a good example of how things aren't always as obvious as you'd think. The original text messaging system was horrible human-computer interaction. It was intended just as a way for engineers to test the connections. You just wouldn't design a real system where people used a numeric keypad to type words: it's terrible! Despite that millions of people used it. In fact lots of people actually send more text messages than they have face-to-face conversations these days. Poor human-computer interaction made wonderful human-human interaction. Why? Because (by accident) texting has just the right properties to be great at what it's used for. It doesn't interrupt you like a phone or even face-to-face chat, so is great for keeping in touch while you are busy. You can also keep several different chat conversations going in parallel. Another plus is that texts are short. Talk to someone and you have to go through a lot of social niceties – you can say goodbye half a dozen times on the phone before you actually manage to put the phone down on a friend! Texts avoid all that.

Splurging sentences

So what is so subtle about human-human interaction? When we talk to people face to face, we don't think up whole messages and then just splurge out the whole thing (as we do in an email). Watch people chatting. Listen for all the umms and ahs, the pauses, the half said sentences, the corrections, the "ok"s and "ah ahh"s from the other person that keep the conversation going. Real conversations don't go...

"This trouble I had in understanding not only what others said to me, but also what I said to them"

They are more like:
You: "This prob...err...trouble I had..."
Listener: "uhu"
You: "in under umm in under..."
Listener: "understanding?"
You: "yes"
Listener: "go on"
...

Meaning in mess

It looks a mess, but it all actually matters to make conversations work. It's about being sure you understand each other. There's more too. Now look for the nods and smiles, the blank faces, turns of the head, the waving hands, the pointing that go with all the "umms" and "ahhs" and interruptions. The listener's interruptions show they are following. Subtle changes in their expression might make you rethink what you were saying. Some of that does come across on a small video image of a person's head and shoulders but not all.

Subtle space

One of the key things missing in a video conference call is 3-D space itself. When people are working together on a problem they use the space around them as part of this subtle communication. A person may briefly glance to another for confirmation: confirmation that comes with a tiny nod of the head. If all you have is a series of images lined up on your screen no-one can tell who is being glanced at in the way you can round a table. Similarly human gestures aren't just about sliding your fingers about on a touch-screen. In deep conversation we make shapes in the air around us with our hands as we talk. Point to that place later in the conversation and people will know what we are talking about, just as it can be clear we mean Jo not Laura when we just say 'she'.

Good, bad (and ugly?)

All this doesn't mean video calls are rubbish. It just means you need to understand what different technologies are good and bad at and use them

appropriately. Email is great for sending information but terrible for anything with emotion – hence the flame wars that result. Phone calls are great for one-to-one conversations, but bad if you just want to communicate a quick message like "Sorry I'm running late". Video calls do let you see each other's expressions and the things you are talking about. They can be good for 2-way conversations. What we are still missing though is a technology that is as good as actually sitting round a table working as a team: a technology that doesn't lose all the messy subtlety of human conversations. Video calls aren't it. Maybe you can invent a technology that is.

This article is based on a keynote talk given by Pat Healey of Queen Mary, University of London, June 2013. (cs4fn was there watching it live!)

A recipe for programming

How is a computer program like a recipe? **Paul Curzon** explains, and as a bonus, tells you how to cook a quick pasta dish.

Programmers are the master chefs of the computing world – except the recipes they invent don't just give us a nice meal, they change the way we live.

Programs are very similar to recipes. They both give instructions that, if followed, achieve something. There is a difference between them, though, and it has to do with language. When chefs invent recipes they write them out in human languages like English. Programmers write programs in special languages. Why's that? It's all about being precise enough to be sure exactly the same thing happens every time. Recipes are often ambiguous which is why when I follow one it sometimes goes wrong. Programs tie down every last detail.

Let's apply some ideas from programming languages to making meals. One of my favourite recipes is a hummus-based pasta dish (see box) so we'll use that.

Hummus and Tomato Pasta

Serves 2

This is a very quick 20-minute after work dish.

Olive oil
1 teaspoon of whole cumin seeds
1 large chopped onion
400g chopped plum tomatoes
200g hummus
150g pasta

1. Add the pasta to a large pan of boiling water. Simmer for 10 minutes.
2. Fry the cumin in the olive oil for a few minutes. Add the onions and fry gently.
3. Stir in the tomatoes and the hummus and leave to simmer for 5 minutes.
4. Drain the pasta and serve.

Structure it!

The first thing to notice about a recipe book is there is a clear structure. Each recipe is obviously separate from the others. Each has a title and a brief description of how it might be used. Each has an ingredients list and then a series of steps to follow. Programs follow a similar structure.

Cookery books use page layout to show their structure. Programmers use language: grammar, symbols and keywords. A keyword is a word that means something special. Once you have decided a word is a keyword you only ever use it for that purpose.

Let's invent a keyword RECIPE to mean we're starting a new recipe. The only time that word will appear in our recipes is to start a new recipe. What follows it will always be the name of the recipe. We will also need to know when the name ends. To make that clear we will use a special symbol made up of open and close brackets ().

We also want to be absolutely sure what is part of this recipe and what isn't. We will use curly brackets: everything between the brackets is part of the named recipe.

```
RECIPE Hummus and Tomato Pasta ()
{
  ...
}
```

No comment?

Recipes usually include a brief description that isn't part of the actual instructions. It is just there to help someone understand when you might use the recipe. Programs have descriptions like this too. Programmers call them 'comments'. Remove the comments and the recipe will still work. We need a clear way to show when a comment starts and ends. We will start them with a special symbol '/*' and end them with '*/'.

```
RECIPE Hummus and Tomato Pasta ()
{
  /*  Serves 2
      This is a very quick 20-minute after
      work dish.
  */
  ...
}
```

Variable storage

What comes next in a recipe is usually a list of ingredients. The idea is to list everything you need so you can have it all ready before you start. I often have a problem following recipes, though, as they don't list absolutely everything. Mid-recipe I might suddenly find I need a frying pan...when mine is crusted with burnt cheese sauce from last night! To avoid that, let's list all the pans we need too. For our recipe we need a frying pan and a saucepan.

Something used to store things (like pans do) in a program is called a 'variable'. Program variables hold things like numbers. The equivalent of the ingredients list 'declares' the variables. Declarations give each variable a unique name used to refer to it and also give each a 'type' – is it a saucepan or a frying pan we need? To be clear about when a declaration ends we add in some punctuation. Programming languages tend to use a semicolon for that – it's a bit like a full stop in English.

```
Saucepan    pan1;
Fryingpan   pan2;
```

This says that in the rest of the recipe when we say pan1 (the variable name) we mean a particular pan: a saucepan (its type). When we say pan2 we mean a particular frying pan.

New assignment

We will make a distinction between things to hold stuff, like pans (variables) and the actual ingredients that go in them: 'values'. We will also follow the TV chefs and start by setting out all the ingredients in little dishes at the start so they are at hand – and make that part of the instructions.

We will need to declare a dish to hold each ingredient, giving its type and giving the dish a name. At the same time we will say what should be put in it before the recipe proper is started. We will use an '=' symbol to mean put something in a variable (i.e., dish or pan). In programs, this action of putting something in a variable is called 'assignment'. So, for example, we will declare that we need a dish to hold the hummus (called hummusDish). We assign 200g of hummus to it.

```
Dish hummusDish = 200g hummus;
```

We are now ready for the recipe proper. We can use assignment as a precise way of moving things from one place to another too. So if we say, for example:

```
pan2  = oilDish;
```

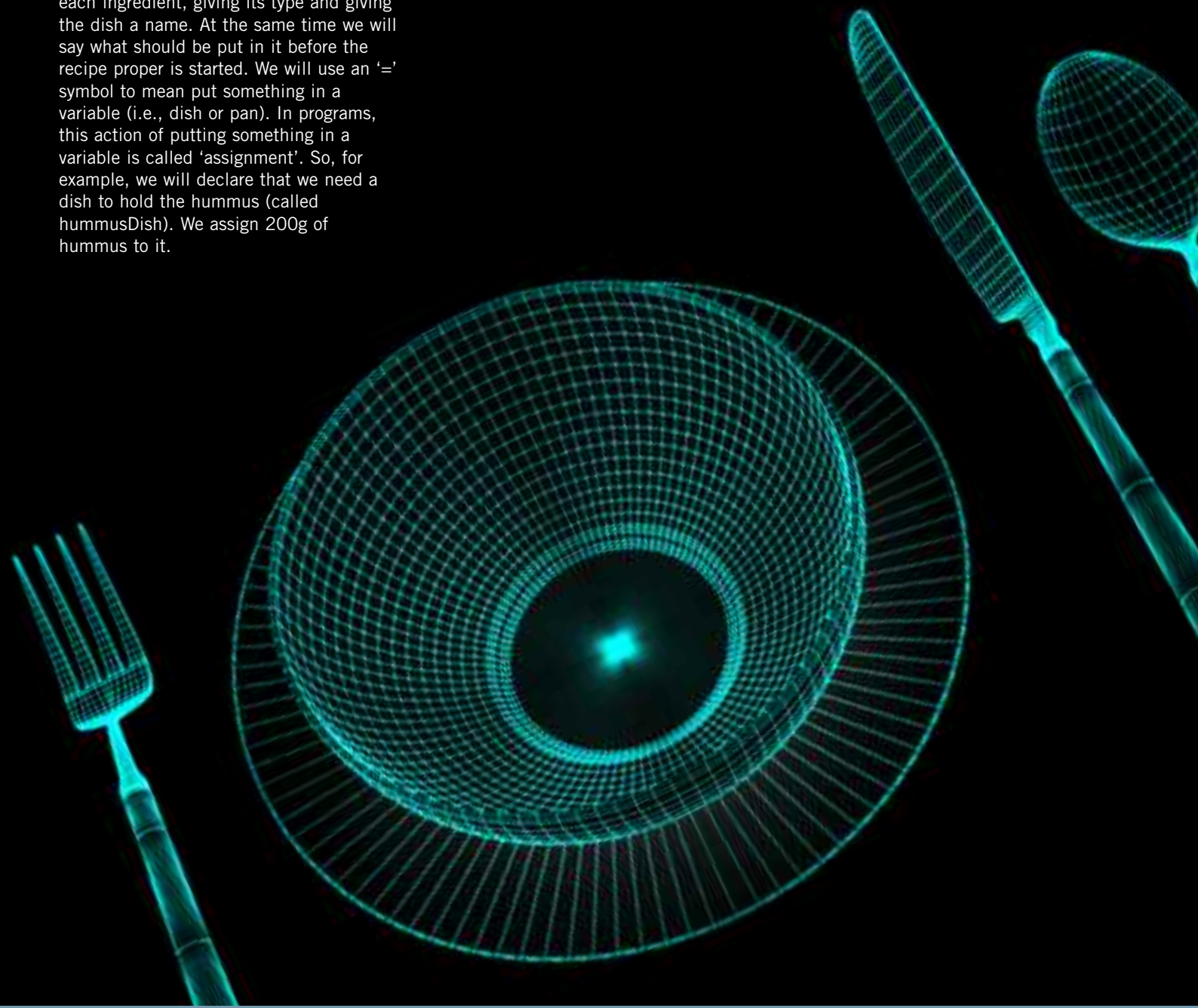
We mean empty the contents of the dish of oil into the frying pan. Programs are slightly different here, as when they do an assignment they don't move things from one place to the other, they copy it. That would be like having a dish that automatically refilled itself whenever it was emptied.

Assignment does NOT move things around, it makes new copies

Often we want to add to whatever is already in a pan. Programmers leave nothing to doubt and say explicitly that is what they mean:

```
pan2  = pan2 + onionDish;
```

This tells us to mix what is in the onion dish with what is in the frying pan, and then leave the result in the frying pan. We will use the + symbol to mean add together and stir. *(continued on the next page)*



A recipe for programming

(continued from page 15)

Methods in my madness

So far all we've done is put ingredients in things and copied them around. To make a meal we need to do various basic cooking things like heat a pan or drain a pan. Rather than spell out every step of how you do that in every recipe, we will use a short hand. We create mini-recipes that say how to do it and just refer to them by name. They are often called 'methods' by programmers. Each is written out just like our recipe. In fact to a programmer our recipe is a method too. When we want to use it we just give its name followed by any extra information needed. For example to heat a pan, we need to know which pan, how high a heat and for how long. We write, for example:

```
Heat (pan1, medium, 12 minutes);
```

This format helps make sure we don't miss something (like the time for example). We need similar methods for draining a pan and serving the meal. We won't give the actual instructions here. In a full program they would be written down step-by-step too and not left to chance.

Time to do it right

We have come up with a language for recipes similar to the ones used for programming. We've used symbols, keywords and very precise punctuation – the language's 'syntax' – to help us be precise. On its own that's not enough – each part of the language has to have a very clear meaning too – the language's 'semantics'. Together they make sure in following a recipe we know exactly what each step involves. There is then less scope for a cook (or computer) to get it wrong. Computers, of course, have no intelligence of their own. All they can do is exactly follow the instructions someone wrote for them (a bit like me cooking).

Here's what our complete recipe looks like as a program.

```
RECIPE Hummus and Tomato Pasta ()
{
  /* Serves 2. This is a very quick after work dish.
     It only takes about 20 minutes from start to finish. */
                        Saucepan  pan1;
                        Fryingpan pan2;

  /* Ingredients */     Dish oilDish = 1 tablespoon of olive oil;
                        Dish cuminDish = 1 teaspoon of whole cumin seeds;

                        Dish onionDish =  1 large onion, chopped;
                        Dish tomatoDish = 400g chopped plum tomatoes;
                        Dish hummusDish = 200g hummus;

                        Dish pastaDish =  150g pasta;
                        Kettle kettle = 500ml boiling water;

  /* Cook the pasta */  pan1 = kettle + pastaDish;
                        Heat (pan1, high, 2 minutes);
                        Heat (pan1, medium, 10 minutes);

  /* Make the sauce */  pan2 = oilDish + cuminDish;
                        Heat (pan2, high, 2 minutes);

                        pan2  = pan2 + onionDish;
                        Heat (pan2, medium, 5 minutes);

                        pan2  = pan2 + tomatoDish + hummusDish;
                        Heat (pan2, low, 5 minutes);

  /* serve */           Drain (pan1);
                        Serve (pan1, pan2);
}
```

Scilly cable antics

Autumn 1869. There were great celebrations as the 31 mile long telecommunications cable was finally hauled up the shore and into the hut. The Scilly Isles now had a direct cable communication link to the mainland. But would it work? Several tests messages were sent and it was announced that all was fine. The journalists filed their story. The celebrations could begin.

Except it didn't actually work! The cable wasn't connected at all. The ship laying the cable had gone off course. Either that or someone's maths had been shaky. The cable had actually run out 5 miles off the islands. Not wanting to spoil the party, the captain ordered the line to be cut. Then, unknown to the crowd watching, they just dragged the cut off end of the cable up the beach and pretended to do the tests. The Scilly Isles weren't actually connected to Cornwall until the following year.

Honky herring

If you think about it, humans are a long tube with legs and arms. We speak from our mouths and the other end of the tube that things come out of, well, that has other uses. But what if emanations from that other end could be used as a way of communication? There is no scientific evidence to date that humans use their bottom to communicate, despite what you might think. But fish do, perhaps. A Canadian study showed that herring seem to use the breaking of wind underwater to communicate with each other, and it wasn't just gas caused by what they had eaten. It's an odd way to communicate perhaps, but could explain why fish don't eat baked beans.

Farming the future

A duck, a horse and a chicken walk onto a farm… It sounds like the start of a bad joke but it's actually the start of a bit of mind mystery as you accurately predict the order your spectator will line up the animals in advance. To dig up more on this fun magic trick, look on the magazine+ section of our website, **www.cs4fn.org**.

The last speaker

The languages of the world are going extinct at a rapid rate. As the numbers of people who still speak a language dwindle, the chance of it surviving dwindles too. As the last person dies, the language is gone forever. To be the last living speaker of the language of your ancestors must be a terribly sad ordeal. One language's extinction bordered on the surreal. The last time the language of the Atures, in South America was heard, it was spoken by a parrot: an old blue-and-yellow macaw, that had survived the death of all the local people.

Why do languages die?

The reason smaller languages die are varied, from war and genocide, to disease and natural disaster, to the enticement of bigger, pushier languages. Can technology help? In fact global media: films, music and television are helping languages to die, as the youth turn their backs on the languages of their parents. The Web with its early English bias may also be helping to push minority languages even faster to the brink. Computers could be a force for good though, protecting the world's languages, rather than destroying them.

Unicode to the rescue

In the early days of the web, web pages used the English alphabet. Everything in a computer is just stored as numbers, including letters: 1 for 'a', 2 for 'b', for example. As long as different computers agree on the code they can print them to the screen as the same letter. A problem with early web pages is there were lots of different encodings of numbers to letters. Worse still only enough numbers were set aside for the English alphabet in the widely used encodings. Not good if you want to use a computer to support other languages with their variety of accents and completely different sets of characters. A new universal encoding system called Unicode came to the rescue. It aims to be a single universal character encoding – with enough numbers allocated for ALL languages. It is therefore allowing the web to be truly multi-lingual.

Languages are spoken

Languages are not just written but are spoken. Computers can help there, too, though. Linguists around the world record speakers of smaller languages, understanding them, preserving them. Originally this was done using tapes. Now the languages can be stored on multimedia computers. Computers are not just restricted to playing back recordings but can also actively speak written text. The web also allows much wider access to such materials that can also be embedded in online learning resources, helping new people to learn the languages. Language translators such as BabelFish and Google Translate can also help, though they are still far from perfect even for common languages. The problem is that things do not translate easily between languages – each language really does constitute a different way of thinking, not just of talking. Some thoughts are hard to even think in a different language.

Time is running out... by the time the AIs arrive, the majority of languages may be gone forever.

AI to the rescue?

Even that is not enough. To truly preserve a language, the speakers need to use it in everyday life, for everyday conversation. Speakers need someone to speak with. Learning a language is not just about learning the words but learning the culture and the way of thinking, of actively using the language. Perhaps future computers could help there too. A long-time goal of artificial intelligence (AI) researchers is to develop computers that can hold real conversations. In fact this is the basis of the original test for computer intelligence suggested by Alan Turing back in 1950...if a computer is indistinguishable from a human in conversation, then it is intelligent. There is also an annual competition that embodies this test: the Loebner Prize. It would be great if in the future, computer AIs could help save languages by being additional everyday speakers holding real conversations, being real friends.

Too late?

The problem is that time is running out. Human standard translation is still a way off. Artificial intelligences that can have real human conversations are an even more distant dream. The window of opportunity is disappearing. By the time the AIs arrive the majority of human languages may be gone forever. Let's hope that computer scientists and linguists do solve the problems in time, and that computers are not used just to preserve languages for academic interest, but really can help them to survive. It is sad that the last living creature to speak Atures was a parrot. It would be equally sad if the last speakers of all current languages bar English, Spanish and Chinese say, were computers.

A funny thing happened on the way to the computer

Laugh and the world laughs with you, they say, but what if you're a computer? Can a computer have a sense of humour?

Computer generated jokes can do more than give us a laugh. Human language in jokes can often be ambiguous: words can have two meanings. For example the word 'bore' can mean a person who is uninteresting or could be to do with drilling ... and if spoken it could be about a male pig. It's often this slip between the meaning of words that makes jokes work (work that joke out for yourself). To be able to understand how human-based humour works, and build a computer program that can make us laugh will give us a better understanding of how the human mind works ... and human minds are never boring.

Many researchers believe that jokes come from the unexpected. As humans we have a brain that can try to 'predict the future'. For example, when catching a ball, our brains have a simple, learned mathematical model of the physics so we can predict where the ball will be and catch it. Similarly in stories we have a feel for where it should be going, and when the story takes an unexpected turn, we often find this funny.

The 'shaggy dog story' is an example; it's a long series of parts of a story that build our expectations, only to have the end prove us wrong. We laugh (or groan) when the unexpected twist occurs. It's like the ball suddenly doing three loop-the-loops then stopping in mid-air. It's not what we expect. It's against the rules and we see that as funny.

Can you tell which joke is the human's and which the computer's?

Artificial intelligence researchers who are interested in understanding how language works have started to look at jokes as a way to understand how we use language. Graham Richie is one such researcher, and funnily enough he presented his work at an April Fools' Day Workshop on Computational Humour. Richie looked at puns: simple gags that work by a play on words, and created a computer program called JAPE that generates jokes.

How do we know if the computer has a sense of humour? Well how would we know a human comic had a sense of humour? We'd get them to tell a joke. Now suppose that we had a test where we had a set of jokes, some made by humans and some by computers, and suppose we couldn't tell the difference? This is called a Turing Test after the computer scientist Alan Turing. If you can't tell which is computer generated and which is human generated then the argument goes that the computer program must, in some way, have captured the human ability.

So let's finish with some one-liners (and a test). Which of the following is a joke created by a computer program following Richie's theory of puns, and which is a human's attempt? Will humans or machines have the last laugh on the test?

1) What's fast and wiry?... An aircraft hanger!

2) What's green and bounces?... A spring cabbage

Find out which was which by going to the magazine+ part of the cs4fn website: **www.cs4fn.org**

Back (page) from the dead

We all know that eventually everyone dies, but it's what we do with our life that counts. Computer scientists are starting to look at ways that they can make death just that little bit different, reshaping the one fixed point in human existence, and sometimes in rather strange ways. RIP can go from 'Rest In Peace' to 'Right, Investigate Possibilities'!

Twitter from beyond

Social networks are a vital part of many people's lives, but what happens when you're dead? Do your tweets need to stop? Does the world need to lose your views? Well no, according to the LiveOn project. It aims to create an artificial intelligence program that learns how you tweet, and the sorts of things you tweet about, like and dislike, and can continue to mimic your online presence even after you depart this earth. As their slogan says, "When your heart stops beating, you'll keep tweeting".

RIP = Replicating Individuals Preferences

Grave situations

When Chinese computer fan Hu Chuang died, his parents decided to commemorate his love of computing in a novel way. His headstone was carved in the form of a computer and monitor. It even included a mouse. This isn't the first time that graves have incorporated a love of technology. There are headstones that carry QR codes you can scan to be taken to a memorial website to find out more about the occupant. But to get even more personal, people have been buried with their mobile phones. In the days before wireless, some were buried with phone lines running to their coffins, in case of premature burial. They never called back though. Wonder why…

RIP = Really, I've Passed?

Dead actors: I'll be back!

Today's computer graphics can help dead actors rise again. Research is being undertaken to generate lifelike synthespians: synthetic actors who look and act like the real thing. Using old movie footage computer systems can learn the shapes of actors' faces, their characteristic movements and even their voice patterns. This process of digital cloning is often referred to as Schwarzeneggerization, after the idea was first introduced in a book where a customer asked that Arnold Schwarzenegger be digitally substituted to replace other actors in classic films. There was, not surprisingly, concern about how these resurrected characters could be used. The widow of dancer Fred Astaire and the Screen Actors Guild successfully lobbied the Californian senate to pass the Astaire Bill, restricting the use of digital clones and giving actors legal rights from beyond the grave.

RIP = Relive Innovative Performances

Google: archive for the afterlife

We live our lives in the cloud. Our photos, blogs, and so on live on the Internet, protected by our passwords. But what happens to all of this intellectual property when we die? Google has thought about this. They are providing a service that will send your login and password details to a chosen person in case your online activity stops for too long, which they consider indicates you are dead. This way someone else can look after all your lovely images and ideas, keeping them safe for the future

RIP = Repurpose Intellectual Property

Speaking of the dead

Alexander Graham Bell, widely recognised as the inventor of the telephone, has found his voice at last, over a hundred years after his death. Bell recorded himself on a wax cylinder reciting lines from Shakespeare and delivering what he may have considered his catchphrase: "Hear my voice, Alexander Graham Bell". The recording method involved speaking loudly into a cone that at the far end had a needle in contact with a rotating cylinder covered with soft wax. The vibrations of the voice, caught and amplified through the cone, caused tiny bumps to be impressed in the wax, capturing the sound but also allowing playback. The cylinder was rotated and the bumps, via the needle, caused appropriate sound in the cone. However wax decays over time and Bell's words were impossible to recover using anything that contacted the brittle wax. The solution employed by the Smithsonian Institution was to use a laser to read the bumps and turn them back into sound, exactly the same technology as is used in DVD players.

RIP = Repeat In Person

cs4fn is edited by Paul Curzon, Jonathan Black and Peter McOwan.
cs4fn is supported by **Google.**

Queen Mary University of London

The Chartered Institute for IT

EPSRC Engineering and Physical Sciences Research Council

COMPUTING AT SCHOOL

For a full list of our university partners see www.cs4fn.org

www.cs4fn.org