PRIMO

Beginning computer programming for kids

An introductory guide to computational thinking and coding for kids aged 3-6 years old.

Plus programming ideas for kids, and the best programming languages for kids.

www.primotoys.com

Table of Contents

Hello

Hello, we are Primo Toys. We're a team of technologists, designers and educators who make toys that help children learn with technology.

You may already know our first toy, Cubetto – a playful wooden robot that helps young children aged 3+ learn the basics of computer programming through hands-on play, without a screen.

We started building it back in 2013, and we made a bit of a splash last year when we launched Cubetto 2.0 on the crowdfunding platform Kickstarter. We raised $1.6m, becoming Kickstarter's most-funded educational technology project ever. At the time there were just five of us in the team. Now we are 17 and, after another Kickstarter campaign we look like this:

We created Cubetto because, like an increasing number of academics and experts, we believe that **coding for kids is a new kind of literacy for the 21st century.**

We think that for girls and boys, all over the world, learning to program will be as important as their ABCs and 123sw in helping them understand the world around them. We believe that introducing children to a world of algorithms, bugs and queues also develops their problem-solving skills, encourages collaboration and nurtures children's creativity.

Like Maria Montessori, the Italian educationalist acclaimed for her method that builds on the way children naturally learn, we think that children learn best through play. We also know that they're motivated by challenges; and that at very young ages, abstraction (the ability to think about a solution to a problem without trying it out first), is difficult.

Coding is a new kind of literacy for the 21st century.

Our aim is to help create a new global standard for coding education that incorporates this and other approaches to the way children learn best, both at home and in the classroom. We want to help a new generation to realise their full creative potential in a world of rapidly increasing technological progress. This ebook is part of that mission.

Why a book and why now?

In the four years since we started, learning software programming for kids – once a niche and nerdy enterprise – has elbowed its way politely into the mainstream. Well-publicised and well-funded private and public campaigns (often helped along by very well-known entrepreneurs and stars – Mark Zuckerberg, Bill Gates, will.i.am...) have helped make this possible. A growing number of innovative designers, researchers, and – of course – crowdfunding platforms, have also helped propel coding toys and programming games to new heights. Children as young as three or four can now use block-based, tangible coding languages like our own or Kibo or Google's Project Bloks; while older children can build and program their own robot kits including Lego Mindstorms and LittleBits. And then there's a growing number of programming languages for kids, like MIT's ScratchJr designed for five to seven-year-olds.

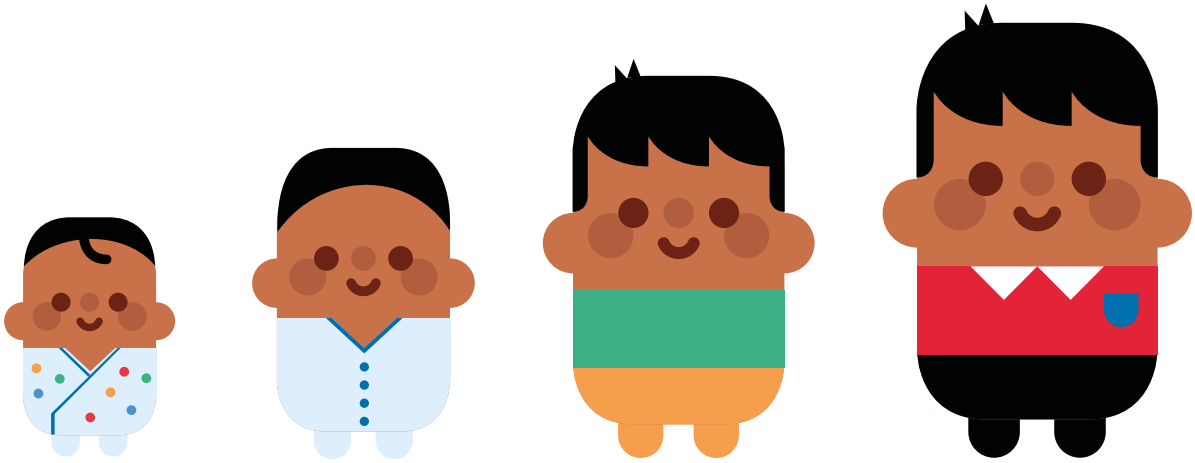
Meanwhile, in some countries including the UK, programming is already part of the curriculum, and there are a huge number of schools that teach coding around the world. So there's never been more information for parents and teachers dedicated to helping to get kids coding. If you Google 'kids' coding ebook' you'll find plenty of engaging publications that offer a step-by-step initiation to a variety of coding languages, all aimed at beginning computer programming for kids.

The problem is that many of these have been written to complement formal schooling, which typically starts in most countries around ages five or six, and have been written for teachers rather than parents. So we decided to produce something that could also help children aged three to six. An introductory guide that brings together all those vital hands-on learning techniques and imaginative tips and tricks we've learned and developed over the years.

Who is this book for?

This book is primarily for parents who are keen to find out more about what computer programming is, and how they can introduce easy programming for kids in a fun and engaging way.

We also hope that teachers, particularly those less familiar with coding and its key concepts,will find it useful both as an introduction and a tool for their lessons.



How does this book work?

In Part One:

'Introducing coding for kids', we dip into some of the key terminology that we'll use throughout the book that typically causes confusion. We'll also ask: 'Why learn to code in the first place?'

In addition, there's a brief but important history of educational coding, which introduces some of the bright minds and stunning innovations that got us to where we are today – it sets the scene for the rest of the book, so please don't skip it!

In Part Two:

We look at two different age groups:

Section i. Three to four years old.
Section ii. Five to six years old.

For both age groups we investigate:

• How children learn.
• Which computational and programming concepts should be introduced at each age.
• What games and exercises (some that use technology, and others that don't) we can use to teach these concepts in fun, authentic and creative ways, as well as programming ideas for kids, and the best programming languages for kids.

In Part Three:

'What next?', we take a quick look at the options available to parents and educators who want to introduce their children to programming beyond tangible devices.

Who will find each section most useful?

While we hope that you'll find each part of this book helpful, Part Two, Section i, on three- to four-year-olds has been written especially with parents in mind. Part Two, Section ii, for five- to six-year-olds has been written with both parents and teachers in mind.

We hope you enjoy using this book as much as we enjoyed writing it.

Part One

What do we mean by introducing computer programming for kids?

Why should kids learn coding and computational thinking at all?

A brief history of coding education for kids

What do we mean by: 'Introducing Computer Programming for Kids?'

Computing terminology can be tricky. 'Coding', 'programming', 'computational thinking' and 'computer science' are often used interchangeably to mean the same thing. To avoid confusion, let's start out by getting them straight.

Computer science is the study of what computers can actually do – it's essentially the theory side of things. Computer scientists test and study what is possible using hardware (the physical components of a computer, like the hard drive and motherboard), and software (the programs and data that run on and that live in a computer's hardware).

Examples, given by Oxford University, of the kinds of things that concern computer scientists include whether computers can help us to model and investigate hugely complex systems like the human body, financial systems or the Earth's climate.

Computational thinking is the thinking tool that computer scientists use for the kinds of investigations given above.

The BBC puts it most succinctly: Computational thinking 'allows us to take a complex problem, understand what the problem is and develop possible solutions. We can then present these solutions in a way that a computer, a human, or both, can understand.'

Computer programming, on the other hand, is the practice of making a computer do things, normally through lines of code that have been written to create an intended outcome: 'If I write ABC the computer will do XYZ.'

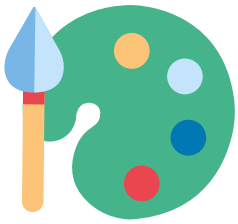
© iStock/RichVintage

Coding, even though it's often used as a catch-all for all three terms above, is just another slightly more contemporary way of saying 'computer programming': ie the act of writing code, normally on a screen, to make a computer do something you want it to do.

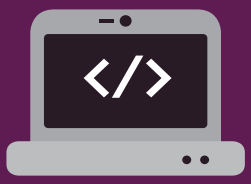
In this book, for reasons we'll outline later, we'll focus predominantly on introducing computational thinking approaches to children in younger age groups (three to four and five to six), tying in some of the basic elements of programming.

For a glossary of all the terms we use in the book, please skip to the back!

Why should kids learn coding and computational thinking?



1) Coding nurtures creative expression



2) Programming demystifies tech

3) It teaches problem solving and persistence

4) Children learn by thinking about doing

5) But children also learn to think about thinking

Why should kids learn coding and computational thinking at all?

There's a common assumption, not always helped by the tech industry itself, that kids need to learn coding because we need more computer scientists. More software engineers, the argument goes, would help shape our digital world for the better and, you know, it wouldn't be so bad for the economy either. (The term 'workforce-ready' comes to mind).

To which the obvious response for a parent or educator is: 'But what if my child or student doesn't want to be a computer scientist?'

While it's true that the big tech firms – Google, Facebook, Apple, Tesla – are snapping up whizzkid engineers, there are far more exciting and compelling arguments for why children should learn to code, beyond: 'Because it's great for GDP.' Here are five of them:

1. Coding nurtures creative expression

Coding for kids is a fundamentally creative process. Just like painting or cooking, with coding a child benefits from the satisfaction – even the exhilaration – that comes from starting with nothing and finishing with something.

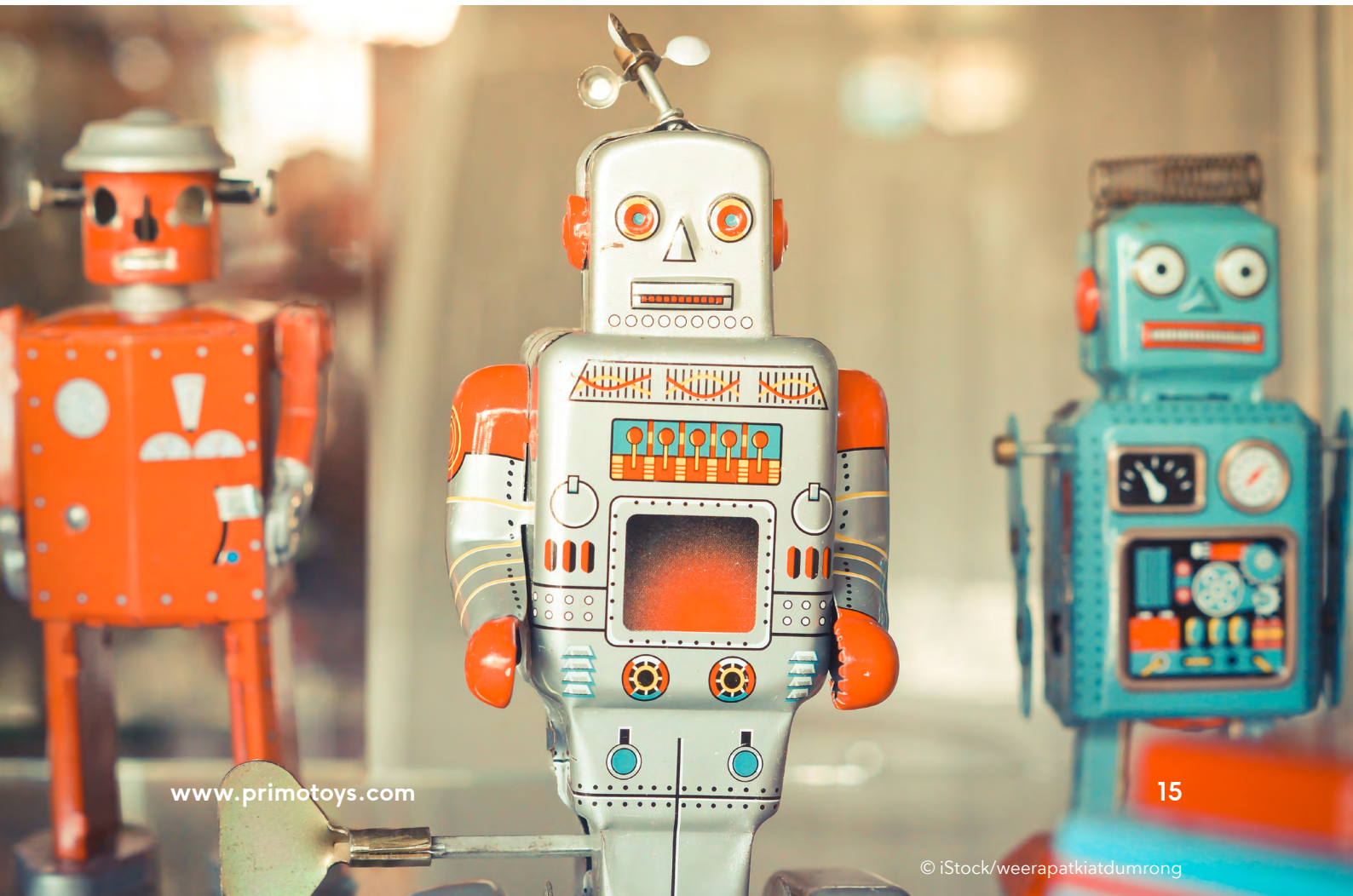
And it goes further. In the real world, creative acts are often limited by the materials we have at our disposal – like ingredients when we cook, or the canvas when we paint. But with programming, where the virtual world is infinite, the only restriction is the child's imagination.

Coding for kids is a fundamentally creative process; starting with nothing and finishing with something.

2. Programming demystifies tech

The University of Oxford forecasts that in the next 20 years as many as 47 per cent of jobs in the United States will become completely automated. Meanwhile, predictions on the number of connected devices that will be in use by 2020 as part of the Internet of Things vary from 20 billion to 75 billion.

Because of this proliferation of devices and computers, there's a growing anxiety about the increasing role of artificial intelligence and computers, in particular whether machines will make workers obsolete. Understanding what computers can and can't do is fundamental in addressing these anxieties. If we can teach children how to remodel the technological world around them, we can help them become creators rather than just consumers of technology.



In the next 20 years as many as 47% of jobs in the USA will become completely automated.

3. Teaching kids programming requires persistence and problem-solving

Anyone who's played with code, from beginners to professionals, will tell you that writing programs can get quite challenging quite quickly. Or put more simply: coding can be frustrating. Really, really frustrating.

This, says computer scientist and educator Sheena Vaidyanathan, is unreservedly a good thing: children 'learn that something doesn't work out but you can quickly fix it and try it again in different ways.'

With an introduction to programming, children learn to think laterally when faced with a problem in coding: 'If A + B didn't work, then maybe A + C will.' Coding also equips kids with the ability to stick with a problem and work on finding a solution.

4. Children learn by thinking about doing

The grandfather of coding education, Seymour Papert (more on him in bit), was a huge advocate of teaching by using programmable robots for kids. He was also a huge advocate of the principle that we learn by doing. As he saw it, the two worked hand in hand: 'Programming the robot to do something helps a child to think about "doing".'

However, to this he added an interesting qualification: 'You learn by doing but you learn better by thinking about what you are doing. I think this is what is most important.'

In essence, thinking about what you want to do, one step at a time, before you do it, enhances the learning process.

With programming, children learn to think laterally when faced with a problem.

5. But children also learn to think about thinking

Papert (him again) also spoke of the discovery and the sense of wonder that children experience when first introduced to programming. 'In teaching the computer how to think, children embark on an exploration about how they themselves think. The experience can be heady: Thinking about thinking turns the child into an epistemologist, an experience not even shared by most adults.'

For us, this is the argument that is the most exciting. More than anything, computational thinking is an unbelievably valuable thinking tool – perhaps *the* thinking tool of the 21st century – and one that can be applied throughout our lives to incredible effect.

A very brief history of coding education for kids

Coding education as we know it today started 60 years ago with a turtle and a four-letter word that sounds like, but definitely isn't, Lego.

First though, let's rewind a little further to the work of Jean Piaget (1896-1980). A Swiss clinical psychologist regarded today as giant of educational thinking, in the first half of the twentieth century Piaget devised a theory called Constructivism, which looks at how learning happens.

Rather than acquiring knowledge, he argued, humans construct it based on past experience and their understanding of the world. Children make sense of their surroundings not as 'miniature adults' or as empty vessels, but 'as active agents interacting with the world and building ever-evolving theories.' Part of this theory included the idea of 'discovery thinking', which asserts that children learn best through doing and exploring.

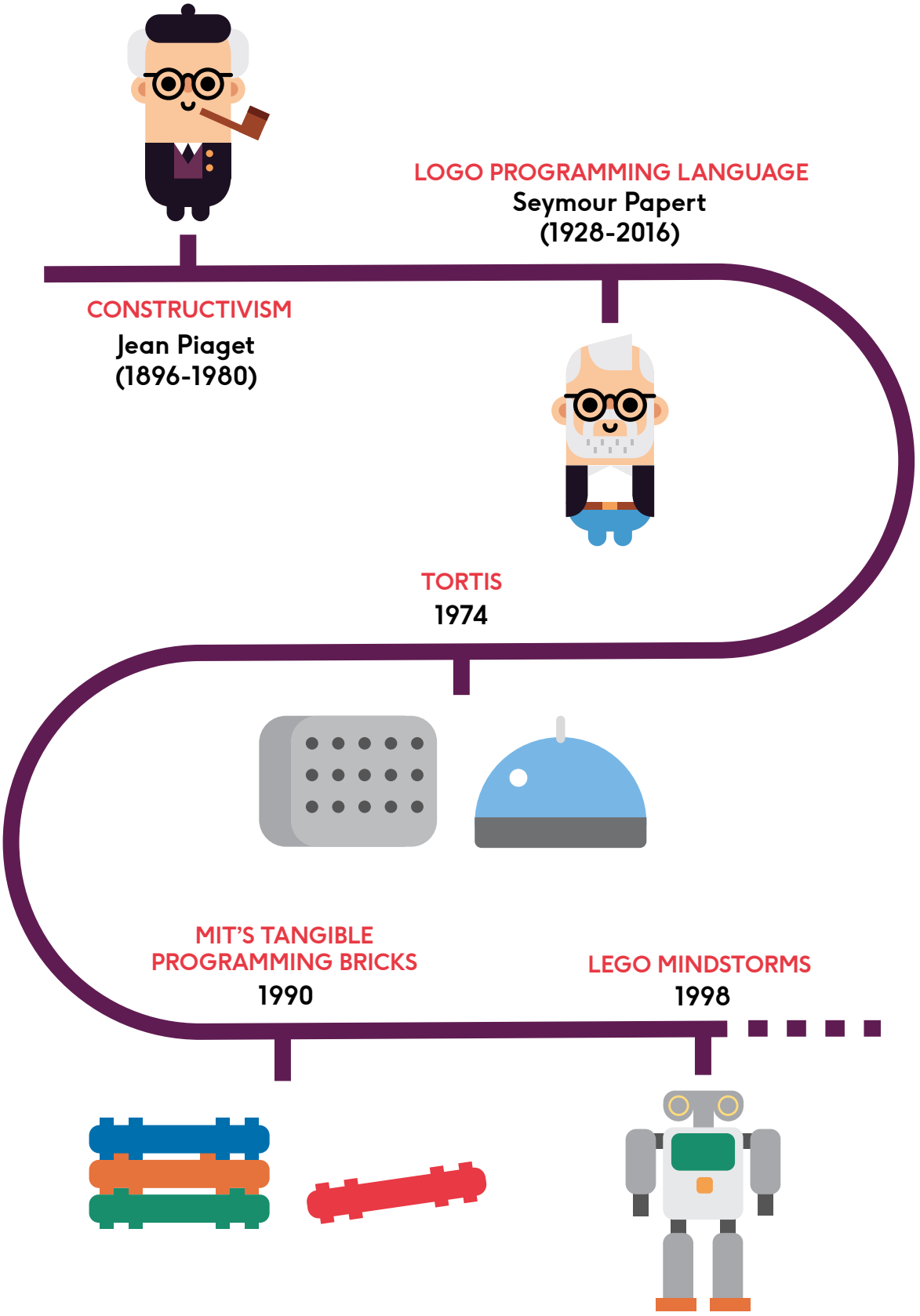
Years later, these ideas found a home, if slightly tweaked, in the work of Piaget's protege – a brilliant South African computer scientist, mathematician and educator called Seymour Papert (1928-2016). In the 1960s while working at MIT, Papert created an ingenious programming language called Logo.

With Logo programming, the child writes on a keyboard commands that produce line graphics, either on screen or in the real world, with a small robot – a 'floor turtle' armed with a pen. With Logo Turtle, for the first time, kids could play with and learn complex programming by producing a screenless, creative outcome via a physical object (in this case a robot).

Logo Turtle programming applies Papert's own educational theory of Constructionism and its four key pillars:

1) Children learn by doing.
2) Tangible objects support concrete ways of thinking.
3) Powerful ideas can empower the individual.
4) Self-reflection helps children clarify their own thinking and connection to the environment around them.

The key principles behind Piaget's Constructivism and Papert's Constructionism were fundamental to further tangible (that is to say 'hands-on') coding technologies, which evolved from the 1970s onwards. These included Tortis, another floor turtle released in 1974; MIT's tangible programming bricks launched in the 1990s;


CONSTRUCTIVISM
Jean Piaget
(1896-1980)
LOGO PROGRAMMING LANGUAGE
Seymour Papert
(1928-2016)
TORTIS
1974
MIT'S TANGIBLE
PROGRAMMING BRICKS
1990
LEGO MINDSTORMS
1998

and of course Lego Mindstorms – the school kits that let kids build and program robots (the third and most recent generation was launched in 2013). The blocks of our own coding toy, Cubetto, can be considered an extreme simplification of Logo Turtle programming language.

We can also see the influence of two other educational greats throughout this period: Friedrich Froebel (1782-1952) and Maria Montessori (1870-1952). Both these figures and their timeless wooden block-based learning materials promoted hands-on, open-ended play in which the child could learn about the properties of the materials with which they played (colour, size, weight), but also about themselves and the world around them.

As I've already mentioned, there is an ever-growing number of toys designed to help get kids coding – ready for a world of robots and AI in which ones and zeros are the way we communicate. It's hard to overstate the importance that the work of Papert, Piaget, Froebel and Montessori still has on the most effective ways of teaching children the basics of computer programming. We will refer to each throughout the book.

Part Two

The use of age groups in this ebook

Section i) Three to Four Years Old
Section ii) Five to Six Years Old

The use of age groups in this ebook

First, a qualification. As any educator, researcher or parent will tell you, children don't develop in unison. At any age there's a wide variation of abilities and skillsets between children of the same age. While some three-year-olds know the alphabet and its sounds back to front, there are others who find the representation of a letter totally alien. The same applies to teaching children to code. Nonetheless, age groups remain a useful heuristic to help guide us, however roughly, to how children learn best at a certain points in their development. Additionally, for most children around the world, formal education doesn't start until the age of five, which is why we have divided this book into a section for three- to four-year-olds and another for five- to six-year-olds.

As the parent or educator, you have a better understanding of where your child or children are developmentally than we ever will, so please approach the following sections as a flexible guide that can be bent, stretched and twisted to their needs.

Section i) Three to Four Years Old

How do children at this age learn?

There's a good reason you hear so many parents talking about threenagers. Children at this age have bundles of energy, tons of self-belief, and don't suffer fools gladly. It helps to play the way they tell you to.

More than anything, three-year-olds just want to do stuff. And even though they may pick up something and then put it down again at dizzying speeds, their interest in the world around them is thrilling and constant. As their speech is still developing, they often find it difficult to articulate what they're trying to do. Which can be challenging for a parent.

So what kinds of cognitive, social and emotional developmental changes are children going through at this age, and what kind of things do we need to consider when teaching them the basics of coding?

Because the research on how we can use technology to teach children is still in its relative infancy, answering this question can be tricky. Fortunately, we were able to speak with and take guidance from the leading academics in the field: Ilene Berson, Professor of Early Childhood, and her husband Michael Berson, Professor of Social Science Education, both at the University of South Florida.

Children don't segment their learning or play

The increasing momentum behind the STEM (Science, Technology, Engineering and Mathematics) and STEAM (which adds Arts to the mix) movements in education, is in part down to a growing belief that the best teaching is interdisciplinary. The argument, as outlined by DT Max in the New Yorker, is that 'similar patterns of thinking underlie many subjects and integrating them makes students smarter. Taking a sculpting class can help researchers understand protein folding; a course in storytelling can aid doctors in communicating with patients. A STEM class in English might use computer algorithms to explore literary style.' Can we apply the same approach to teaching coding education? Absolutely.

A STEM class in English might use computer algorithms to explore literary style.

‘Children don’t segment their brains into “Now I’m going to be a scientist and now I’m going to be a mathematician.” They weave in and out of literacies and activities seamlessly,’ says Ilene.

While two children are playing a block-sorting game with round and square blocks, they might be building their spatial awareness, honing their social-emotional skills, and also developing their ability to problem-solve. This has interesting implications for teaching computational thinking, suggesting that a more blended approach to teaching the subject, where skills are learnt in combination with other seemingly non-related topics and knowledge, is more beneficial than a prescribed ‘coding session’.

Causality is starting to develop

'They're beginning to understand that "if" I do this, "then" this will happen.' says Michael Berson. These lessons are being learnt all the time, often in the rough and tumble of playtime: "If I hit my friend, then they will push me back".'

The aim of the adult is to nurture the exploration of causal thinking, while at the same time helping to prevent any negative outcomes, like, say, a fight with another child. How to do this? Using tangible coding toys is a low-risk method of encouraging this type of thinking. 'It creates a safety net,' says Michael. Children think: 'If this happens and I don't like it, no big deal, I can just change it.'

Hands-on learning trumps screens

Why is hands-on learning, or kinaesthetic learning, so beneficial at this age? It's not just about removing the distractions of screens. Ilene highlights the sensorial qualities of hand-held items. 'When you have a two-dimensional representation of something, like on a book or a screen, your sensory interaction with that thing is limited to sight.'

Something that can be held, on the other hand – let's say a wooden block or a piece of tangible code – can also be touched, tasted (if the child decides to bite it), smelt (compare wood to plastic, say), and heard (if the child decides to bash one item with another). Consequently, handheld items provide richer sensory information to the child.

Very young children use a lot of physicality in their learning process.

The nature of hands-on learning experiences also benefits the child. For Dr Sarah Gerson, an academic at the University of Cardiff's Psychology Department, the pros of hands-on learning over screen learning are that: 'The social and interactive nature of hands-on experiences makes them more engaging and richer sources of information.'

There's more. Ilene says that: 'Very young children in particular use a lot of physicality in their learning process. Through movement they are embodying knowledge. You can see that when young children engage with tangible technologies, they become an extension of the technology itself.' We see this with three- to five-year-olds when they're programming robots, like our own Cubetto, when they'll often act out what they want to do first before writing the program. So in effect, when children turn, spin, jump and bump they become a code-learning resource in themselves.

The social and interactive nature of hands-on experiences makes them more engaging than screen learning.

It's better to let kids negotiate their own way through an activity.

Play is great, but open-ended play is better.

'There's a tendency with adults to step in and get in the way, which can inhibit the child,' says Ilene. She adds that from a developmental point of view it's better to let children negotiate their own way through an activity, especially when that negotiation involves other children.

There's no question that this can be can be a messy process. The child, for example, won't always follow linear steps. Rather than going down the usual A-B-C route, they may try to go from A-C and then they realise that they needed to go via B in the first place. But when they have to figure things out together, they see a more enhanced skill set,

says Ilene. Rather than directing the child, the role of the parent should be that of a facilitator, asking good questions to help the child along.

In her research for Tortis (that stands for Toddler's Own Recursive Turtle Interpreter System – an early floor turtle that children could control by use of a buttons on a control panel), published in 1974, Radia Perlman (another star of educational coding and design), lamented how overbearing parents could be during her sessions with children:

'Parents were especially bad since they were anxious for their child to appear smart and consequently nagged at the children to go onto new things and yelled at them for giggling. Parents were a help when they had the right attitude, however, suggesting projects, laughing with the kids, playing, with them, etc.'

Moreover, by being overly prescriptive, says Professor Marina Umaschi Bers (one of tangible coding's leading thinkers), parents risk sheltering the child from what the renowned computer scientist Alan Kay calls 'hard fun' – something that is both enjoyable and challenging. By working on their own and setting themselves challenges, children manage their frustration and also develop 'confidence in one's ability to learn.'

Indulge their sense of agency

'People talk about the terrible twos, but three- to four-year-olds,' smiles Ilene, 'typically have the sense of a great deal of agency too. Normally by four to five years old we see children becoming socialised so that this agency is lost.'

The benefits of this self-belief as Ilene sees them are clear: less-inhibited creative thinking, and a willingness to try things out and make mistakes.

'Part of the challenge for adults is to nurture that agency without things becoming too chaotic,' she says. A more child-centred approach to learning (as outlined in the point above), with light touches from the parent or teacher, can help this.

Abstraction is difficult

'The ability to abstract – making symbolic representations of things – is more challenging the younger children are,' says Ilene. 'This is because their language skills and and their experience are so much more limited.' A typical three-year-old, for example, has only half the life experience of a typical six-year-old. Children at this age rely on concrete examples of things to inform how they see the world and draw conclusions. Because abstract thinking is removed from the 'here and now', children naturally struggle.

What should we be teaching children at age three to four?

It's a good question. In 2014 the Economist ran a story on coding education that highlighted the problem of where to start with coding for kids: 'Even basic matters, such as striking the right balance between conceptual exercises like the sorting game and actually writing programs, are still not settled.'

Given everything we've addressed above – the benefits of tangible play and interdisciplinary learning, the willfulness of 'threenagers' and their difficulties with abstraction – it's unlikely that most very young children will benefit from jumping headfirst into early lessons on Python, JavaScript, Scratch or any other on-screen coding language. Like most things, it helps to start at the beginning when learning computer programming, and that means computational thinking.

What do we mean by computational thinking?

At the beginning of the book we defined computational thinking as a way of thinking that lets us break down big, complex problems into smaller ones. These problems can then, say, be divided between a team of individuals (such as children in a playroom), according to their different skills and abilities.

But in reality there's no formal definition of what computational thinking is. In fact, most definitions are, necessarily, pretty broad and encompass a range of different concepts. There's also no consensus on which concepts should be counted as part of computational thinking and which shouldn't. If you have a moment, check out the differences between the Google, Harvard and Barefoot Computing (content accessible to registered teachers) interpretations, which are significant. Each outline serves a different purpose for a different audience.

For this book, we've borrowed most from the language and model of computational thinking devised by Barefoot Computing – a fantastic online resource set up to help UK primary school teachers (who teach four- to 11-year-olds), with computer science, and has also informed our thinking for a large part of this section – albeit with our own tweaks here and there.

So without further ado, here are the concepts and ideas that we think are most developmentally appropriate for children aged three to four.

Algorithms

First off, algorithms.

An algorithm is a set of rules that define a sequence of operations. Often when we think of algorithms we think of them in terms of computers, ie a series of instructions that tell a computer what to do. Famous algorithms that dominate our everyday lives include Google Search, Facebook's News Feed and the 'You may also like' suggestions on Amazon.

However, algorithms aren't restricted to computers. They can also be written for humans and everyday activities, for example getting dressed in the morning or making a cup of tea. The steps and rules that we follow for these processes determine whether we achieve success or failure. Of course, in order to work effectively, the rules or instructions on which an algorithm runs must be clear and unambiguous. If they are not, the algorithm malfunctions or breaks.

Algorithms aren't just restricted to computers; they can be written for humans in simple, everyday processes too!

Logic

Logic is also something we can start introducing to this age group. Logical reasoning is a way for us to understand and explain why things happen the way they do.

For example: If I'm at A and I want to reach C, do I have go via B or can I skip it entirely? Logic is important because, unlike people – who change their mood throughout the day, along with how they work – machines are predictable. Logical reasoning allows us to understand how machines work, and in turn what we can do with them.

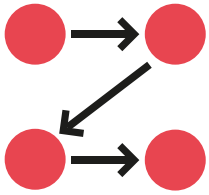
Alongside these concepts, we can also start to introduce certain different approaches to computational thinking.

Tinkering

Tinkering is all about trying things out and learning by trial and improvement. Young children naturally want to explore and experiment. This approach nurtures that instinct.

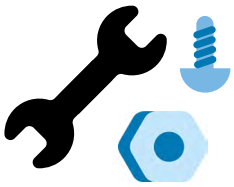
What we should teach children at age 3-4

Algorithms

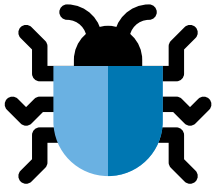


Logic

Tinkering

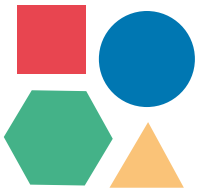


Debugging

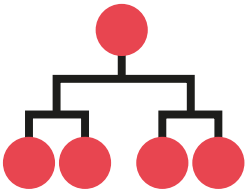


What we should teach children at age 5-6

Abstraction

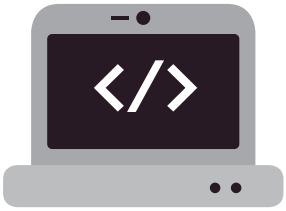


Prediction

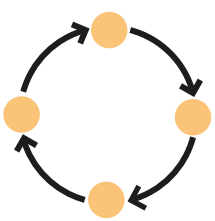


Sequencing

Programming



Repetition

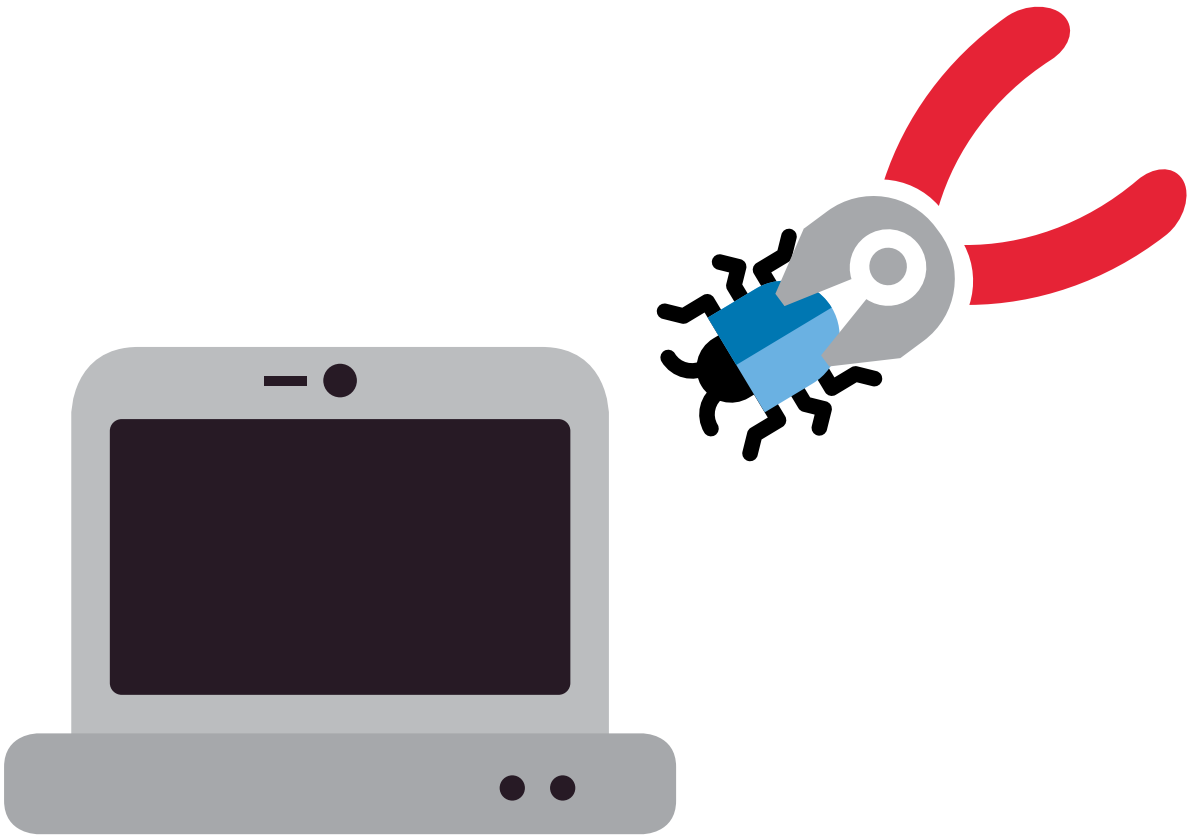


Debugging

Then there's debugging, a technical term that describes a process of literally getting rid of the 'bugs' (or mistakes) within an algorithm. Debugging is a process that we use to varying degrees of success all the time in real life – it's a way to find out why, for example, a new toaster keeps burning the bread, and once we know, how to set it right.

Barefoot Computing have this handy debugging recipe:

- Predict what should happen.
- Find out exactly what happens.
- Work out where something has gone wrong.
- Fix it.



Activities for teaching these concepts

So, what activities are best to introduce these concepts and approaches to children aged three and four? In this section we recommend a few of our favourites.

Unplugged activities

Because computational thinking is a way of thinking – a process by which we can understand a complex problem, and in turn understand the different ways that that problem can be solved – **it doesn't require using computers or screens at all.** Instead, we can use a variety of playful 'offline' or 'unplugged' games and activities. (For the record, the notion of 'unplugged' activities was popularised by the team at CS Unplugged, whose website is a repository of fantastic of free learning activities that take place, of course, off-screen).

Because computational thinking is a way of thinking, it doesn't require using computers or screens at all.

1) Leaving the house routine

Why

Real-life routines are a simple and engaging way to introduce algorithms. Leaving the house, because it's basic and regularly repeated, is a good place to start.

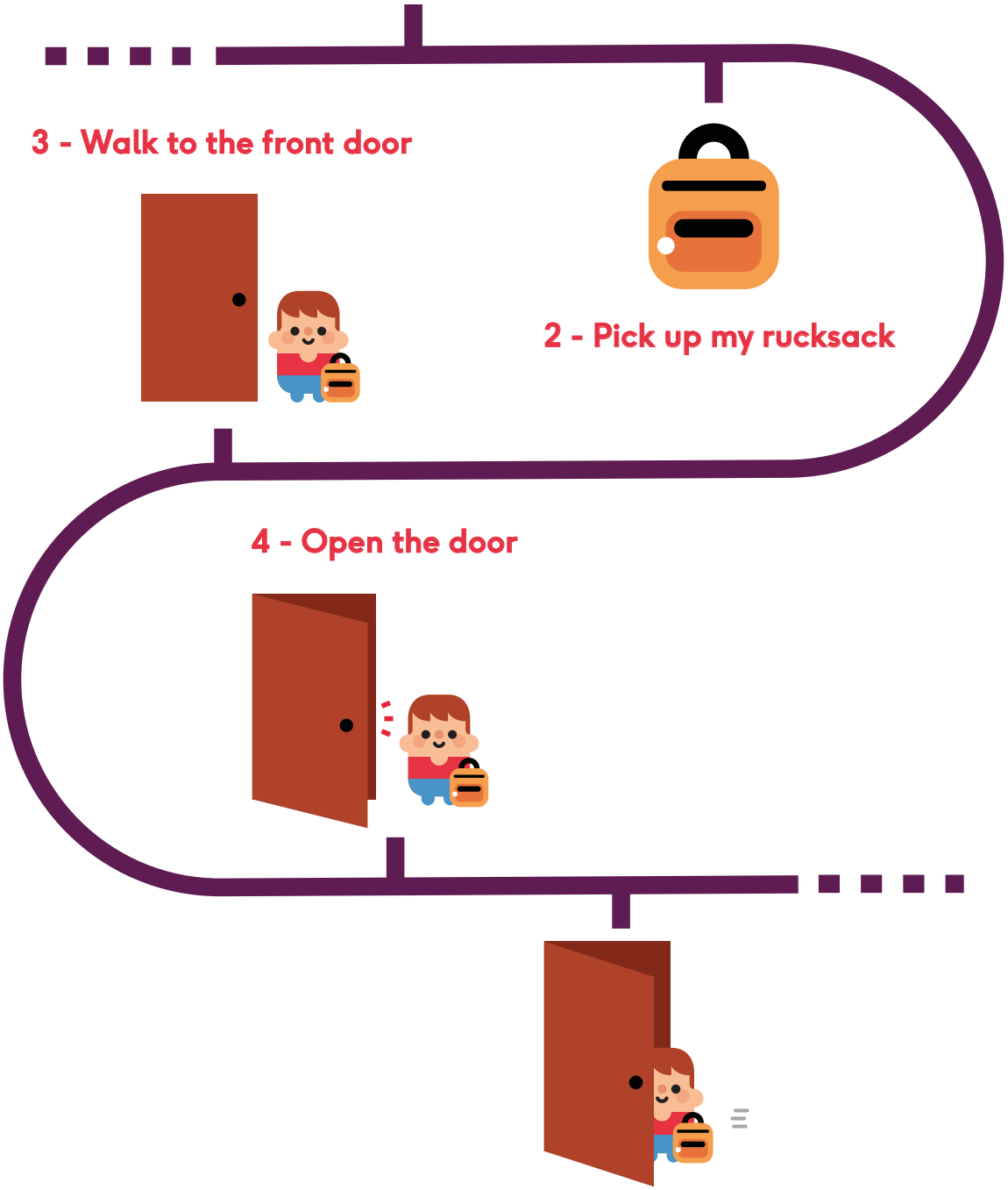
The activity

Begin by familiarising the child with the steps we take when we leave home, along with the terminology for those steps. You can do this by vocalising the process every time you go out.

For example:

First I pick up my keys
Then I pick up my [wallet / purse / rucksack]
Then we walk to the front door
I open the door
You [the child] can walk through the gap
Then I walk through the gap
I close the door behind me
Finally I lock the door

1 - Pick up my keys



2 - Pick up my rucksack

3 - Walk to the front door

4 - Open the door

5 - Walk through the gap

Continue to say these steps aloud until the child knows them. Then try to encourage them to begin saying them at the same time as you. Once they're doing this comfortably, ask them to begin vocalising the process on their own.

At this point, it's fun to start start asking them questions: 'Why do we do the process in this order? Can we change any of the steps and still have the same effect?' Test their hypotheses (eg 'You can pick up your wallet before your keys' or 'You can walk through the door without opening it'). They'll soon see which parts of the process – or algorithm – they can adapt and change, and which they can't. Remember, it's good to keep a constant dialogue going so that the child is explaining their thinking throughout.

You can also make the algorithm more detailed as the child becomes more confident and familiar with the process. For example: 'I put my hand on the doorknob; I twist the doorknob; I pull the door towards me.'

Other ideas

More everyday opportunities to explore algorithms include:

- Getting out of bed
- Getting dressed
- Going to the shops

In fact, any routine that has a clear order of steps that are repeated regularly (daily) can work well.

2) Programmable Parent

Why

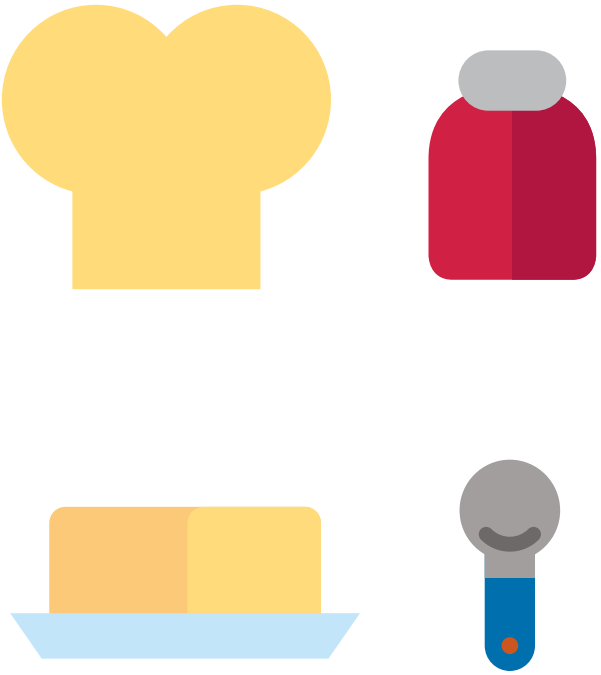
This neat exercise, like the one above, can also be applied to a variety of real-life scenarios. Alongside algorithms, children also get a chance to see action causality (if I do this, then that will happen) with satisfying immediacy. They also put logic into practice, as well as tinkering and debugging.

The activity

Tell your child that they're going to help you make jam on toast. Show them a piece you've made earlier and tell them the aim of the game is to make it look exactly the same.

Tell the child to imagine that you are a robot and that they have to give you instructions (or write the computer program) on how to make the jam on toast. (This could be a neat, creative opportunity to really bring the activity to life – and drive home your new robotic alter-ego – by making props with the child, such as a robot head out of papier maché for you to wear, or a 'control button' that the child has to press every time they want you to complete a separate step in the process.)

Using your best robot voice, begin by having a conversation with your child about what you need to make the dish. Elicit from them the ingredients (jam, bread, butter) and the utensils (knife, plate,



toaster, bread board). Ask how you make toast, to the level of encouraging them to show you where in the kitchen the toaster is and demonstrating how it works.

Once they have the fundamentals ask them to tell you what to do: ‘Please give me your instructions, master!’, or something similarly robot-like.

Most likely they’ll make early mistakes like asking you to put jam on before you make the toast, or putting jam on before the butter. Exaggerate these mistakes, while also making it playful and fun, so that the child becomes aware that it’s difficult to do one thing before another. It’s up to them to debug the process on their own. Meanwhile, throughout the exercise it’s up to the parent to gently coax the child

toward clear, unambiguous instructions of, say, three to four words. Remember to keep this informal – a conversational approach is best. Logic is also important here. For example, 'Can I cut the toast on a plate rather than a chopping board and still get the same effect, while saving a bit of time and washing up?'

Designing an algorithm like this is likely to take a few attempts, and quite a lot of tinkering, to get right. Possible outcomes include the child getting frustrated or angry, or conversely not caring at all. The point is that if it's made to be fun over a number of attempts the child will come to see that clear instructions can set up a positive and creative outcome.

(To see an example of this activity on film, click here.)

Other ideas

The programmable parent can be adapted to fit a variety of situations:

- Making cereal in the morning
- Turning on a story CD
- Buying food in the supermarket, and so on.

For a larger, more creative version of this activity, the programmable parent can play a part within a treasure hunt. Work with the child or children to design a treasure map of their own, being as imaginative as possible. It's up to them to direct you around the garden. Just as with making toast, there are plenty of opportunities to encourage them to think logically and engage in real-time debugging, with the added benefit of rooting their understanding of these procedures in other areas of knowledge such as nature and storytelling.

A treasure hunt encourages kids to think logically and engage in real-time debugging.

Simon Says

Why?

If you grew up with Simon Says, you'll know what a simple game this is. But its simplicity is deceptive, because it's a terrific way to smuggle in some key computational thinking ideas and principles.

What?

In case you've never played it before, Simon Says is a game for a minimum of two players. One player (in this case the adult) takes the role of Simon, who issues instructions to the other player or players. These are often active, silly, fun or all three at once, for instance: 'jump up and down', 'touch the floor', 'make a funny face.' However, these instructions should only be followed by the child if they're prefaced with the words: 'Simon Says.' In larger groups, if a child executes an instruction that doesn't start with 'Simon Says,' they're out.

What is learned when we include an activity is that the instructions have to be unambiguous, just like in an algorithm. So unless there is a 'Simon Says' before the actual instruction, then the algorithm or the 'code' itself is broken because it stops being logical. In addition, the adult has a verbal tool that allows him or her to insert false data into algorithm, either by not including the 'Simon Says' command, or by using a muddled command like 'Says Simon' or 'Stephen Says.' Consequently the child needs to engage in debugging throughout.

Other ideas

Simon-Says-style commands can give added zest and complexity to the Programmable Parent and Leaving the House games. In addition, the child can also take on the role of Simon, which gives him or her the opportunity to experiment with unambiguous commands, as well as trialling what it feels like to consciously create a program or algorithm with bugs in it.

Plugged in activities

The benefits of 'unplugged' activities are obvious. They are easy to set up, quick to play and, of course, free. However, as a company that makes a wooden robot designed to teach kids the basics of programming, it would be remiss of us if we didn't say that certain tech-based (let's say 'plugged in') activities are greatly beneficial too. We are followers of Seymour Papert's Logo turtle after all.

If you are using a tangible programming toy, the child needs to first be introduced to the product and how it works. In the case of Cubetto, introduce the Control Board as a sort of remote control that children can use to send instructions to the wooden robot after blocks are inserted to make a sequence of commands.

Without the board, there is no way of sending Cubetto his instructions. Meanwhile, in the case of, say, Kibo, the child must understand that they need to scan the barcode of each instruction block using the barcode scanner on the robot.

We mention this because it is important for children to understand that each device is only able to move with a human's command. This is not only empowering, but also key to understanding computing and computer programming.

4) Left, right, forward, backward

Why?

The language of direction is fundamental to our lives, but it's also fundamental to various tangible coding languages for kids that use a robot – Cubetto included. By introducing these terms (left, right, forward, backward) we can help the child understand this terminology, while also introducing them to sequencing using precise, unambiguous instructions.

The activity

The terms 'left', 'right', 'forward' and 'backward' are introduced to the child by the adult. This can be pre-taught in earlier offline activities mentioned above. Indeed, the adult could help the child make them more precise, like 'a quarter turn right', 'a quarter turn left' or 'backwards one step'.

When the children are familiar with these instructions, Kibo, Cubetto or another robot powered by tangible code can be introduced. The corresponding directions for the buttons or (in Cubetto's case) blocks are pointed out to the child. Then, using a map, the teacher sets a square as an end goal or a 'home' – the aim is to reach this square.

Next, it's up to the child to narrate what they're programming the robot to do, as they do it. There is of course no right or wrong way for the child to move the robot to get home, that's up to them. The role

of the adult is to guide the learner and help them with the correct terminology as they're narrating which way the robot is moving.

This activity comes with a variety of challenges. The child must first understand that the robot needs to be guided to a destination using a program. It can't simply be picked up and put there. In addition, it's likely that 'up' and 'down' may be used by the child to mean 'backwards' and 'forwards'. The child may also struggle to understand initially that the way Cubetto is facing requires different instructions to the way the child is facing (just as they might find in earlier Programmable Parent games).

However, the real challenge is for the adult to avoid interfering too much when the child is programming the robot. In particular the parent should avoid worrying about the sophistication of the child's algorithms. Even for many five-year-olds, programs limited to two or three steps (or blocks) are perfectly normal. This should inform where the 'home' square on the map is set.

Other resources

It's important to emphasise that these should not be seen as comprehensive guide or curriculum, but rather examples of the kinds of activities that we've found are most beneficial.

Here is a selection of online hubs (including our own) where you can find more great activities and ideas for teaching children to code, focused on ages three to four.

Primo Toys - A wealth of Cubetto-themed stories, activities and video tutorials that parents and teachers alike will find handy.

Barefoot Computing - Mainly focused on educators, including more on Barefoot's breakdown of computational thinking and programming, along with activities to support them.

CS Unplugged - Also teacher-focused, but shouldn't be missed by any parent looking for off-screen games and puzzles to teach computer science.

Kibo - More on Kibo, the research behind it, and a selection of step-by-step tutorials on how to use it.

Section ii) Five to Six Years Old

How do children at this age learn?

Between the ages of five and six, from a learning point of view, much of what we discussed in the previous chapter still applies. Multiple literacies are still developing in tandem; interdisciplinary learning based on lived experiences still works best; and open-ended, hands-on play remains beneficial.

Typically, however, we see three important changes: two cognitive, one environmental.

1) Children are beginning to learn how to abstract

It's around this time that a child can begin to think symbolically, which is essential in order to teach kids to code. A younger child, says Early Childhood Professor Ilene Berson, may want to throw a wooden block against a wall because they really have no idea how that block will behave when it hits the wall until they try it out. Will it explode on impact, bounce back or fly straight through the wall? By the time a child is five or six, they know that the block will ricochet off the wall or shatter, making a noise and – in the process – probably putting any adult nearby on edge.

In general, children in this age group have more lived experience at their disposal – they have a clearer idea of how the world works because

they've seen more of it. Consequently, they can begin to make sense of the world using their imaginations, to plot mental routes and to extrapolate ideas without necessarily needing to enact it first in the real world (not that physical reenactment can't still be useful).

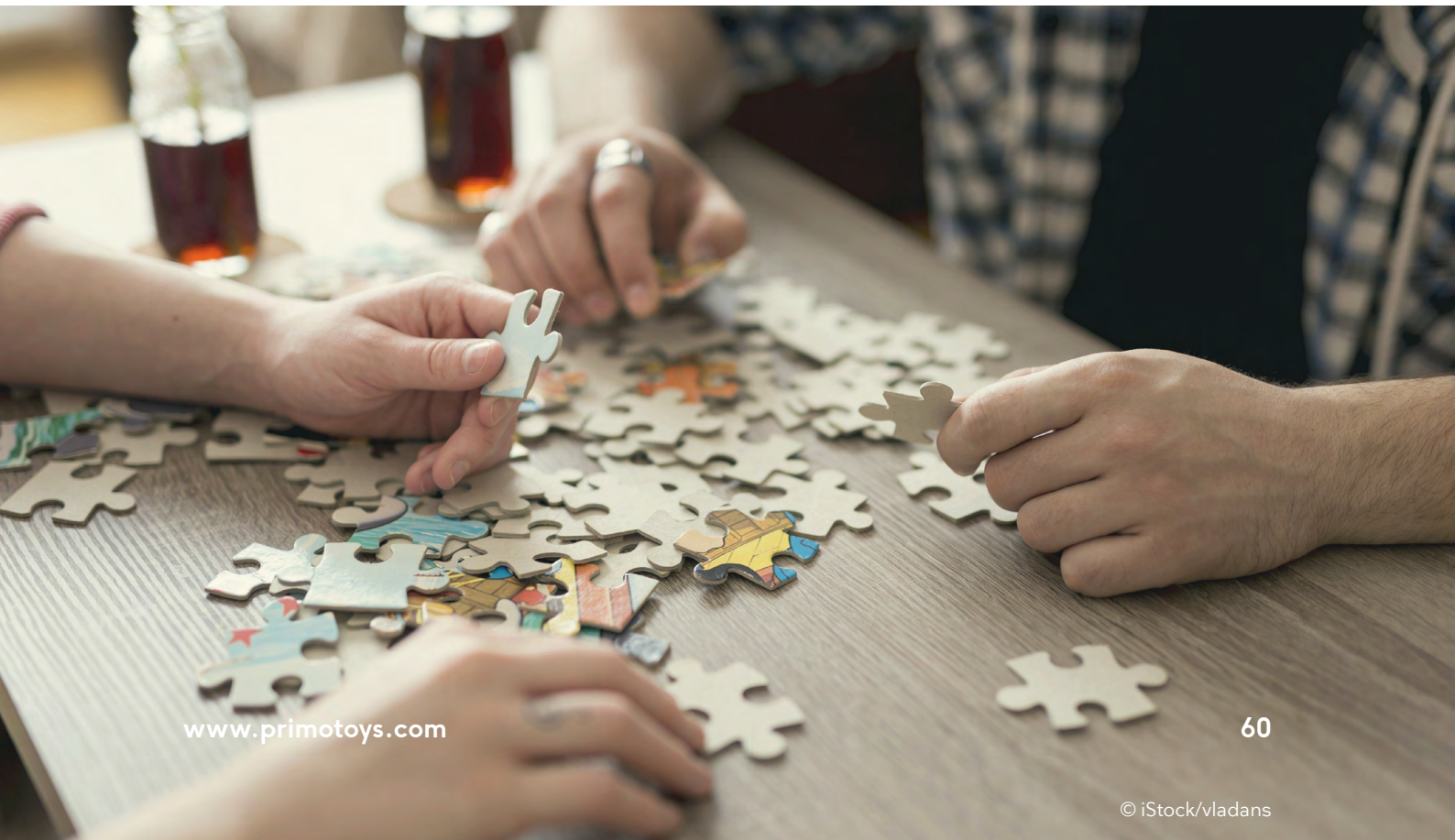
As an interesting side note, the notion that children can abstract at this age runs contrary to psychologist Jean Piaget's stages of cognitive development, which for decades were a mainstay of thinking among educators. For Piaget, it was only once a child had reached the age of 11 (the so-called formal operational stage), that they could think in logical terms using symbols. There is now evidence to suggest that we've underestimated children's capabilities drastically. A 2016 study found that **children as young as three to four were able to grasp algebraic concepts.** This is significant because, of course, algebraic thinking is all about symbolic representation.

Recent research shows that children as young as three to four were able to grasp algebraic concepts.

2) Growing patience and persistence

A child's approach to activities and problem-solving in this time also changes. Following on from the willfulness of the terrible twos, says Ilene, at around three and four years old children begin to develop an intentionality when they play – they identify a goal and work towards that goal. Of course, in open-ended play, the goals can shift and transform quite quickly, but they're still there. This behaviour lends itself directly to beginning computer programming for kids.

In the earlier years of schooling at around ages five and six, children begin to direct their intent towards reaching a goal using solution-driven processes: 'How am I going to get this done?'. At the same time the attention that they can devote to a particular task (before frustration sets in) grows too, and so does their perseverance and ability to collaborate with other children. Like a big bag of tricks, says Ilene, these social-emotional and cognitive strategies can be used for coping with problems and other people.



With open-ended play, children develop resilience and perseverance.

3) Children are enrolled in formal education

The move from the informality of the home to the more formal setting of the classroom is one of the most significant moments in a child's life. Think of the changes: a new pedagogical environment outside the home or preschool; extended and routine learning and playtime with peers in the form of the school day; and a new person – the teacher – who takes on most of the responsibility of the child's learning.

Of course, within a school setting, we also start to see a more formalised approach to learning, and in many cases a movement away from the kind of open-ended play that children may have enjoyed previously at home and in preschool.

‘Among the benefits of open-ended play are that children start developing a sense of resilience and perseverance,’ says Ilene. ‘If something doesn’t go the way they intended, they can say: ‘“Oh well, that didn’t work,” then go back and try it again differently. But what happens as they move up the year groups or as they get older is that we often see a movement towards a more prescriptive way of learning, specifically one where there’s a right and and a wrong answer.’

The problem with socialising learners around a ‘2 + 2 = 4’ or ‘right answer, wrong answer’ approach from such a young age, says Social Science Professor Michael Berson, is that it it detracts from their willingness to experiment. A wrong answer is a piece of negative feedback, and at this age when children still have so much to learn there are often more wrong answers than right. The greater the amount of negative feedback a child receives, the less likely they are to ‘give it a go.’

What should we be teaching them at age five to six?

Building on the computing concepts that children have already learned as three- to four-year olds (logic, algorithms and so on, as we describe Section i), remains fundamental. In this respect, it’s helpful to think about teaching computational thinking as a little like teaching mathematics. You begin with addition, then you learn repeated addition (or multiplication), and then repeated multiplication (or indices). In each case we’re adding complexity to that early base knowledge.

Part of this is just naming the concepts with which they’re already familiar. It can be as simple as sounding out the words ‘al-gor-ithm’ or ‘lo-gic’ when an algorithm is being built or we’re using logical

thinking. Or if the aim of a particular part of a class is to debug a program, we need to make clear that this is 'debugging' and so on. The aim is to familiarise children with the language of programming so that it starts to inform their thinking.

There are, however, a few new coding concepts that we can introduce to children at the ages of five and six to help them further understand computer programming.

Abstraction

What is abstraction exactly? In the words of Jeannette M Wing, the renowned computer scientist, abstraction is a process that allows us to identify which 'details we need to highlight and which details we can ignore.' It's a way of filtering out patterns and specific details to get to the bare essentials of something, so that we can then create a representation of that thing.

Let's clarify this a little further. Say we want to create a representation of a dog that is true for all dogs. We begin by looking at what makes a dog dog-like. Dogs have common characteristics, like four legs, two eyes, two ears, a tail that wags and the ability to bark. They also have characteristics that are specific to each individual dog, like fur colour, length of fur, colour and size of eyes; some dogs may yap and others bark, while some may almost never bark at all. To create a basic abstraction – or representation – that is true of dogs in general, we need to group together these common characteristics and disregard those characteristics that are specific to each dog. (To see this example but with cats, check out the BBC's Bitesize page).

We intuitively make and use abstractions to make sense of the world – take for example a TV guide or a metro map. In the case of the TV guide we need to know which programmes are being shown and when, but we don't need to know the name of every actor who stars in it or every step of the storyline. For the metro map, we need to know the order of stations on a particular line, and even where different lines cross, but we don't need to know the distance between each station or whether the carriages are old or new. In both cases information is stripped out to clarify understanding.

Abstraction is important because it helps us to solve problems. Without it, we may arrive at the wrong conclusion. Take our dog – if we base our abstraction on specific, non-common characteristics, we may end up thinking that all dogs are made up of shapeless mounds of dark brown, shaggy hair, that emit loud barking sounds.

Prediction

Prediction, which fits under the banner of abstraction, also becomes important at this age. Can a child look at the piece of physical code, or even hold a series of commands in their head, and have an idea of where the programmable robot (or human) will end up? If they can, they are able to solve multiple problems in their head where previously they would have used more time to enact their solution in real life, in real time.

Programming

Excitingly, at around this time we can also begin to introduce programming to kids. Put simply, programming is the expression of the ideas of computational thinking (algorithms, decomposition, logical reasoning), in written form (coding). Of course, this code doesn't have to be on the screen but can be in the form of tangible code too, using a coding toy.

Sequencing

One of the key ideas within programming is sequencing. When broken down, a program is a collection of sequential instructions. These instructions must be clear and unambiguous otherwise the program won't work.

Repetition

Repetition, in which the execution of an instruction or instructions within a program is repeated, is also important. Most programs will have repetition built in, both as a means for saving time (it beats assembling the same line of code over and over again), and making it easier to read.

Activities to teach these concepts

Depending on the level of the class, the teacher may choose to kick off with a few lessons that bring children up to speed, for which those activities outlined in Section i would do well. Of course, these can be tweaked accordingly. We've always liked the idea of a kind of aerobics class where, rather than exercises, the teacher calls out directions as a means to revisit left and right and forwards and backwards commands. Again, these activities can pave the way for tangible coding activities down the line. Once the children are happy with two or three block sequences using their coding toys, try the following activities.

1) Find home

Why?

Alongside an opportunity to play around with prediction and abstraction, three different learning styles are catered for in this simple activity.

What?

A large grid is drawn on the classroom floor or playground in chalk. A home square is chosen by the child and confirmed with the teacher (or if it's at home, the parent). In Round 1, the child is invited to act out the code they'd need to use to arrive at the home square.

In Round 2, the same child is then asked to listen to a set of instructions given by another child or a teacher, and asked to identify on the map where this will take them.

In Round 3, the child is asked to visualise a path to the home square using home-made instruction cards (ie square cards with arrows and commands), before programming another child. (Alternatively the child could also program a robot like Kibo or Cubetto. For Kibo the child simply needs to arrange a program of wooden instruction cubes, scanning the barcodes of each, and pressing the button when they want Kibo to execute a line of code. For Cubetto, the child assembles the relevant coloured instruction blocks within the Control Board. When they are satisfied that this combination is correct, they just hit the Go button to watch the wooden robot move to the right square – hopefully! – on the map).

Not only does this exercise encourage the use of unambiguous instructions and then debugging if the sequence of instructions is incorrect, but abstraction and prediction is introduced by asking the child to visualise the route in Rounds 2 and 3. Moreover, it's also an opportunity for the teacher or parent to see which of several learning styles might suit a child most: visual, auditory or kinaesthetic. Round 1 is kinaesthetic, allowing the child to walk around and interpret the code themselves; Round 2 is auditory, where the child has to make a prediction by listening; and Round 3 is, naturally, visual.

Other ideas

Of course, rather than programming another child in Round 2, the child could instead program a robot. Once the children are happy with programming the robot for a route of, say, four blocks, the teacher or parent can also add in another level of complexity by restricting the number of instruction blocks or steps the child can use. In order to complete the route, the child is forced to use repetition (in the case of our coding toy, Cubetto, this means using the function line, while with another robot like Kibo it requires the use of repeat blocks).

The parent or teacher can also do exactly the opposite, and provide an overly long piece of code to get to the home square. It's then up to the child to make it more efficient.

2) Coding and Storytelling - a robot adventure

Why?

Storytelling provides a vivid and engaging framework to introduce, explore or practice programming principles with children. Conversely, we can also explore the composition of stories using a computational mind-set.

The activity

This activity uses Cubetto or Kibo or any other robot powered by a tangible coding language. It also makes use of Cubetto's gridded, themed maps (which come as part of the the playset), although homemade maps will also work well.

The teacher opens by telling a well-known story, for example The Tortoise and the Hare. The teacher then invites the children to discuss how we come up with stories and who writes them – the answer to aim at is 'an author.' The teacher then tells the children that today, they're going to be authors too and that they're going to write a story using coding to help. The teacher sets Cubetto down and invites the class to give directions.

For every new map square that Cubetto lands on, the students have to dream up and write down a new part of the story. So, the final journey of Cubetto – let's say from the mountains to the sea – is literally based on the route Cubetto took using the code written by the class.

There are several elements of this activity that commend it. Children get to create their own stories while simultaneously writing the code for the robot itself – one feeds the other. In addition, children are able to learn something about the topics they might discover en route, for example how a mountain is formed, where rivers and streams come from, or the benefits of freshwater over saltwater for humans. It's another example of a creative and holistic approach to coding.

Other ideas

There's plenty of room to extend the reach of this particular activity. Teachers could, for example, encourage their coder-storyteller students to tell their stories to children in different classes of the same age, or even learners in the year below. This way, they share their stories and a new learning framework that can extend around the school or even at home: 'I wrote this story for you with my Cubetto.'

Other resources

Just as with the previous section, these activities shouldn't be seen as exhaustive. Here we outline a selection of online resources for more computer programming and computational thinking activities, games and ideas for kids.

Primo Toys - A selection of Cubetto-themed stories, activities and video tutorials that parents will find handy too.

Barefoot Computing - Aimed at educators, this has the full outline of their signature breakdown of computational thinking and programming, plus activities to support them.

CS Unplugged - Teacher-focused, but shouldn't be missed by any parent looking for fun and enlightening off-screen games and puzzles to teach computer science.

Kibo - More on Kibo, the research behind it, and a selection of step-by-step tutorials on how to use it.

BBC Bitesize - This colourful and engaging revision guide for the UK's Key Stage 1 and Key Stage 2 levels of the computing curriculum could be invaluable if you're a teacher in Britain. However, its clear language and summaries will benefit any international educator, along with parents too.

Code.org - For US teachers, Code.org's K-5 curriculum is jam-packed with developmentally appropriate lesson plans.

Part Three

What next?

What next?

Over the previous pages we've outlined a particular approach to introducing very young children to coding. One that starts with computational thinking and its core concepts; that favours hands-on learning, open-ended play and child-centred activities; and that takes kids away from the screen to fit the pedagogy of educational greats like Maria Montessori, Friedrich Froebel, Jean Piaget and Seymour Papert.

This book has been very much an introduction, and a concise one at that. As a parent or teacher, you may be thinking: 'How can I build on what the child has learnt over the course of this book? How can they explore coding and computational thinking from the ages of seven upwards?'

Well, there are different routes for different styles of learning. We talk through some of the options below:

1) Continue learning with tangible coding

When children hit seven years old, tangible coding doesn't suddenly become redundant – there is still very much a space for tangible code in the classroom and at home. Cubetto's instruction board, for example, has trillions of combinations, and a variety of fiendishly difficult (and enjoyable) puzzles can be created with its instruction blocks. At a recent workshop we held at Google's London offices, even adults struggled when we removed Cubetto's forward instruction blocks. The tech firm's employees had to use brain-busting levels of abstraction to solve a coding challenge we set them.

Storytelling also comes into its own at this age. Children can begin to draw their own new and larger maps with details of their own choosing. They can craft new props (think jet packs or back packs) for their robot coding friend. They can even use colouring pens to expand the world of Cubetto for new adventures. This has two handy effects: firstly, it provides more creative and cross-curricular opportunities to root programming in art, geography, history, science and so on. Secondly, by building and growing the world of their robotic friend, children can really own the story they've created, and in turn their own learning process. It's a neat way of personalising coding education for the child, by the child.

Cubetto can be used to devise fiendishly difficult (and enjoyable) puzzles.

2) Start to think about on-screen coding

When children are around the ages of six or seven, you may also want to start thinking about introducing junior screen-based coding languages. Scratch, a free programming language designed by MIT for eight- to 12-year-olds, and its younger sibling ScratchJr, designed for five- to seven-year-olds, have both become popular in recent years. In both cases, the emphasis is on creativity and learning through coding, within which the child can make their own interactive games and stories. If the young learners are especially talented and motivated, even starting them out on popular everyday programming languages like Python shouldn't be overlooked.

Programming languages such as Scratch and Python can provide a great introduction to on-screen coding.

3) Make robots

Sometimes, when we talk to parents about the kinds of learning that kids can do from seven years old and onwards, we hear them create a (false) dichotomy between tangible and on-screen coding. There's a tendency to see it as one or the other. But actually, there's no reason why we can't combine an on- and off-screen approach just as Logo Turtle did 60 years ago. We're talking, of course, about robotics.

By building with and programming robots, children gain an understanding of both hardware and software as well as STEM subjects. They can create in the digital world, and still benefit from the real-world interactions and skills that come from robotics projects – negotiation, problem-solving, engineering, collaboration, patience and persistence. For these reasons, we believe that robotics kits

like Lego Mindstorms become especially effective and engaging educational resources towards the end of early childhood.

The DIY process at the heart of robotics also eventually opens the door to an entire community of fellow makers of all ages, cultures and countries. Maker Faires around the world are rich with opportunities to collaborate on new projects and make friends with people you may never have met otherwise.

We speak from personal experience. We grew out of the maker community, and over several years and many iterations, we designed a robot that is now used in more than 20,000 schools and homes around the world. We've found that even in adult life there are few things more challenging, more exciting or more rewarding than making and programming your own robot.

Of course, you will know your child or children best, so it's up to you to work with them to find out which direction they want to go. Our hope is that this book gives you and them a running start.

Happy programming!

Glossary

Abstraction - Using abstract representations (things that don't exist in the real world), to formulate ideas, solve problems and come to conclusions.

Algorithm - A set of unambiguous instructions that perform a specific task.

Causality - The understanding that an action will produce an effect.

Coding - A contemporary term for computer programming.

Computer programming - The practice of making a computer do things through a sequence of instructions, which are often written in code.

Computational thinking - A fundamental skill that anyone can learn, which helps to identify and break down complex problems so that they can be solved, either by a computer or a human.

Computer science - The study of what computers are actually capable of doing.

Concrete thinking - A way of thinking bound to facts and the here and now (for example by way of physical objects that can be held and seen).

Constructionism - A learning theory developed by Seymour Papert, which sprang from constructivist principles but also argues that learning is most effective when people construct tangible objects or artefacts in the real world.

Constructivism - A 20th-century learning theory, closely associated with Jean Piaget, which proposes that humans construct knowledge based on their experiences, rather than acquiring it.

Hands-on play - A style of play that advocates the use of physical, hand-held objects and often, but not always, avoids the use of screens.

Kinaesthetic learning - An approach to learning which advocates learning through physical activities rather than, for example, through books and lectures.

Logic - A method of reasoning that helps us understand why something behaves in a certain way.

Logo - An educational programming language developed by Seymour Papert, Cynthia Solomon and Wally Feurzeig at MIT in 1967.

Montessori - An educational approach developed by the Italian educator Maria Montessori, which places emphasis on hands-on play and child-centred learning.

Open-ended play - A style of play in which children can express themselves and explore freely without being restricted by preset limitations from adults, or games with a predetermined conclusion.

Programming - See ‘computer programming’.

STEM - An acronym that refers to the academic disciplines of Science, Technology, Engineering and Mathematics. The STEM movement promotes more effective integration of these disciplines in the classroom.

STEAM - As with STEM but with the inclusion of Art and design too.

Tinkering - An approach to making and problem-solving through trial and error.

Bibliography

Books and eBooks

• Tim Bell, Ian H Witten and Mike Fellows, Adapted for classroom use by Robyn Adams and Jane McKenzie, 'CS Unplugged' (2015)

• Jason R Briggs, 'Python for Kids - A Playful Introduction to Programming' (No Starch Press, 2013)

• Helen Caldwell and Neil Smith, 'Teaching Computing Unplugged in Primary Schools' (2016)

• Seymour Papert, 'Mindstorms', (Harvester Press, 1980)

Journals and research

• Carl Benedikt Frey and Michael A. Osborne, 'The Future of Employment' (Oxford Martin School, 2013)

• Marina Umaschi Bers and Elizabeth R. Kazakoff, 'Put your robot in, put your robot out: sequencing through programming robots in early childhood', J Educational Computing Research Vol.50 (2014)

• Marina Umashi Bers, 'The TangibleK Robotics Program: Applied Computational Thinking for Young Children, ECRP v.12 no.2, (2010)

• Marina Umaschi Bers, Louise Flannery, Elizabeth R. Kazakoff and Amanda Sullivan, 'Computational thinking and tinkering: Exploration of an early childhood robotics curriculum' Computers and Education (Elsevier, 2013)
• Paulo Blikstein, Arnan Sipitakiat, Jayme Goldstein, João Wilbert, Maggie Johnson, Steve Vranakis, Zebedee Pedersen, Will Carey, 'Project Bloks: designing a development platform for tangible programming for children' (2013)
• Anthony Ginn, 'Interview with Seymour Papert', Practical Robotics (1984) Extract here
• Michael Horn and Robert J.K. Jacob, 'Tangible Programming in the Classroom with Tern' (ACM Press, 2007)
• Steven Kurutz, 'One Big Workbench', New York Times (2013)
• Seymour Papert and Cynthia Solomon, 'Twenty things to do with a computer', Educational Technology Magazine (Englewood Cliffs, NJ, 1972)
• Radia Perlman, 'TORTIS - Toddler's Own Recursive Turtle Interpreter System' MIT AI Memo No. 311/Logo Memo No. 9 (MIT AI Lab, Massachusetts Institute of Technology, Cambridge, MA, USA, 1974)
• H. Raffle, A. Parkes, and H. Ishii, 'Topobo: a constructive assembly system with kinetic memory. In Human Factors in Computing' (ACM, 2004)
• Jeanette M. Wing, 'Computational thinking and thinking about computing' Royal Society (2008)

News articles and blogs

• '21st Century Literacy: New Initiative Makes the Case that Learning to Code is for Everyone,' Berkman Klein Center for Internet and Society at Harvard University (2014)
• 'Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015,' Gartner (2015)
• 'A is for Algorithm', The Economist (2014)
• Paulo Blikstein, 'Seymour Papert's Legacy: Thinking About Learning, and Learning About Thinking' Transformative Learning Technologies Lab Standford Univeristy
• Dan Crow, 'Why Every Child Should Learn to Code', The Guardian (2014)
• Tony Danova, 'Morgan Stanley: 75 Billion Devices Will Be Connected To The Internet Of Things By 2020' Business Insider (2013)
• George Dvorsky, 'The 10 Algorithms That Dominate Our World', iO9, (2014)
• Anya Kamenetz, 'Coding Class, Then Naptime: Computer Science For The Kindergarten Set' NPR (2015)
• DT Max, 'A Whole New Ball Game', New Yorker (2016)
• Saul McLeod, 'Jean Piaget', Simply Psychology (2009)
• Dave Munger, 'A Simple Toy, And What It Says About How We Learn To Mentally Rotate Objects', Science Blogs (2008)
• Derek Thompson, 'A World Without Work', The Atlantic (2015)
• Sheena Vaidyanathan, 'Computer Science Goes Beyond Coding', EdSurge (2015)
• Jeanette M Wing, 'Computational Thinking: What and Why?', The Link (2011)
• Mark Ylvisaker, 'Concrete vs. Abstract Thinking' LEARNnet (2008)

Videos

- 'Code: The New Literacy' (2013) YouTube
- 'Introducing Project Bloks' (2016) YouTube
- 'Program Your Teacher to Make a Jam Sandwich (Sandwich Bot) Junior Computer Science', 2012) YouTube
- 'Young Programmers Think Playgrounds, Not Playpens | Marina Bers | TEDxJackson' (2015) YouTube

Websites and online resources

- Barefoot Computing
- BBC Bitesize (KS1 Computing)
- CODE.org
- Computational Thinking with Google
- Computing at School
- Concept to Classroom
- Dr Techniko
- Gov.uk, National curriculum in England: computing programmes of study'
- Kinder Labs
- Scratched
- University of Sydney
- Wikipedia, 'Kinaesthetic learning'
- Wikipedia, 'STEM'
- Wikipedia, 'STEAM'
- Wikipedia, 'LOGO'
- Wikipedia, 'Constructivism'
- Wikipedia, 'Jean Piaget'