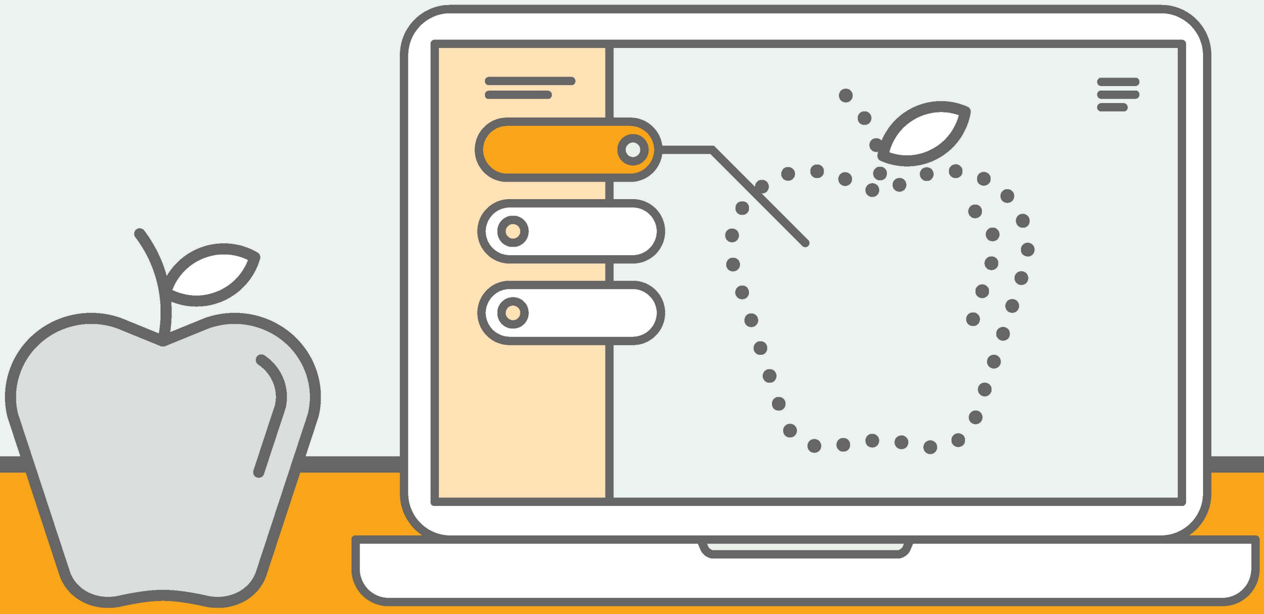
teachers learning code



DIGITAL TOOLBOX:

Quick Start Guide

An educational program by

CANADA LEARNING CODE

Table of Contents

Welcome

WE WANT TO INSPIRE CANADIANS

through coding education to become empowered digital citizens who can understand, participate, and shape our country and the world as creators and innovators of technology.

Whether you are a teacher in a classroom, a program coordinator at a community centre, a homeschooling parent or a Girl Guide troop leader — we've put together this guide to help you teach anyone in your community how to code.

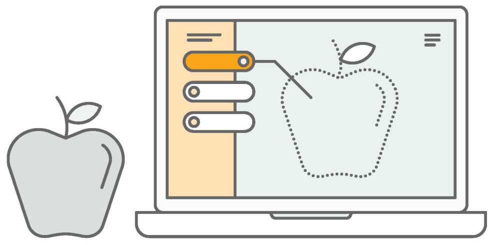
After running programs for several years, we have learned a thing or two about teaching people how to code. In our series of Teachers Learning Code guides, we share many of our tips and tricks for getting starting, resources to familiarize yourself with, and lots of easy–to–follow, and even easier to implement, coding lessons to empower the future generation of technologists across Canada.

We encourage you to use as much or as little of our resources as you like, to remix the lessons we've created, or to create new ones and share them with our community. Our guides should serve as your jumping off point — where you choose to take your program is the fun part and is entirely up to you!

Technology is creative — have fun!

AN INTRODUCTION TO

Teachers Learning Code



What is Teachers Learning Code?

Using technology to change the world through teamwork, creativity, and of course, code.

Teachers Learning Code is a program designed by Canada Learning Code, a national charity focused on inspiring kids to be builders — not just consumers — of technology through coding lessons and challenges.

Teachers Learning Code is an approach to introducing coding that can be scaled back or expanded depending on the needs of your group of learners. Our programming can be facilitated by non-technical or technical educators and youth program managers within schools and community groups.

Our series of 'how-to' guides and coding challenges are designed for K–12 learners. They were all created with the core elements and features that our programs are known and loved for.

> teacherslearningcode.com

How Teachers Learning Code was Developed

At Canada Learning Code, we've taught thousands of youth across the country through our Girls Learning Code, Kids Learning Code, and Teens Learning Code workshops, camps, and after-school programs. Our content has been developed in partnership with educators and industry–leading experts and tested, time and time again, with youth between the ages of 3–17 inside and outside of the classroom.

For Teachers Learning Code specifically, we took this methodology and curriculum development process one step further and engaged non-technical educators in the planning, development, and execution of the material. We hosted focus groups, sent out surveys, co-taught with teachers in schools and community centres, observed our materials being used in the classroom and iterated, iterated, and iterated again — all with the ultimate goal of developing a program that works for students and educators alike.

teachers learning code

OUR FORMULA FOR

Successful Programming

Our content has been designed specifically with girls in mind and with the goal of creating gender-balanced tech environments.

Research has shown us that girls approach the computer as a "tool" useful primarily for what it can do; boys more often view the computer as a "toy" or an extension of the self. At Canada Learning Code, we don't only teach "how" to build, but also "why" it is important. We offer an opportunity for youth to learn about technology and collaborate together.

Fun and engaging content
We LOVE technology and want to pass it on. With ~~the~~ developers, designers, and professional educators, we've created content that not only showcases beginner–friendly concepts, but also highlights ideas to be excited about.

The opportunity to learn and collaborate alongside like-minded students
Through hands–on team–based exercises kids will feel part of a resourceful group who are all curious about technology, and will find learning about it both intriguing and rewarding.

We're all about confidence and empowerment
Teachers Learning Code lessons and challenges are all about instilling the growth mindset in learners. We're all about teaching youth how to learn from their mistakes and developing confidence and empowerment through learning and building.

Digital Toolbox

The Teachers Learning Code Digital Toolbox is a series of guides, lessons and resources that have been designed to give educators the general skills, knowledge, and confidence to introduce coding to their students.

The guide you are reading right now is our Quick Start Guide and includes high-level information on how to bring coding to your classroom. This guide is designed to be used in conjunction with our tool-specific guides that cover the basics of our favourite coding tools, like Scratch, Mozilla Thimble, and Mozilla X-Ray Goggles. Our tool-specific guides also come with detailed lesson plans and coding activities ready-made for classroom delivery!

TIPS FOR TEACHING CODE & HAVING FUN WHILE DOING IT

1 Ensure you are familiar with the tool, but don't worry about being an 'expert' — allow your learners to teach one another.

2 Have a clear vision for what you want to accomplish and find a champion in your school to co-teach with you! It is all about integrated learning.

3 Have a growth mindset or 'fail-forward' approach. Spread the belief that abilities are not dictated by talent alone, but can be developed through hard work and perseverance.

4 Bring outside experts in. Invite guest speakers and volunteers from the community to lead mini lessons and be there as extra coding support.

5 Let your learners and their creativity be your guide. What do they want to explore more? What do they want to learn?

6 Be creative, don't be afraid to fail, and most importantly, have fun!

TEACHING CODE FOR

Beginners

What is Code?

The simplest explanation for code is that it represents a set of instructions that are given to a computer in order to execute a certain task. Together, these instructions create an algorithm.

Computers take direction extremely literally, which means that any instructions in your code/algorithm should be precise and specific. For example, when reading a PB&J recipe (an algorithm in itself!), a human would likely understand that "Put the jam on the bread" is direction to spread some amount of jam on a slice of bread. A robot, however, might interpret this entirely differently. Should the robot put the jar of jam on top of the bread bag? This might seem silly to you, but you will soon see how literally a computer will take direction.

A clearer set of instructions for a computer might be:

1. Turn the lid on the jam counterclockwise until it is completely loosened.
2. Lift the lid off the jar of jam.
3. Place the lid beside the jar of jam.

...and so on.

Code is extremely versatile — we can use it to control robots, build webpages, create video games, analyze large datasets, and more! The possibilities are endless. While we use different coding languages to complete different tasks (ie. HTML & CSS for web-building, R for data analysis, Javascript for flying robots!), each of these languages is really just a different way of communicating your instructions or algorithm to the computer.

Why Teach Coding

Technology is everywhere and it's not going away. Science, technology, engineering and math (STEM), especially when integrated with other disciplines, are the skills of the future. We want to equip Canadian youth with the critical skills they need to navigate the world we live in today and thrive in the future.

But why teach coding?

According to the Information and Communications Technology Council of Canada, there will be an estimated shortage of more than 200,000 ICT workers in Canada by 2020. Learning to code can lead to rewarding and lucrative careers for our youth. That said, teaching kids to code is about more than just helping children understand the technology they are using and secure employment in the future. At a fundamental level, it improves problem-solving and critical thinking skills.

We think it's important for youth to learn to code for a few reasons:

Coding is a superpower. Learning to code let's kids build — not just consume — the technology around around them, from video games to websites, robots, and more!

Coding helps kids develop new ways of thinking. Learning to code helps kids develop other crucial and transferable skills — like computational thinking — allowing them to tackle problems outside the realm of coding in new and innovative ways.

Coding helps kids understand the world around them better. If we teach biology and mathematics in order to understand the world around them, then knowing the basics of how computers communicate and how to engage with them should be a given.

Coding can help change the world. Learning to code empowers kids to use technology as a creative tool to build solutions to challenges people face everyday.

Coding is fun! We want kids to experience the satisfaction and thrill of building something of their very own.

Diversity Matters

It's a fact that the technology industry has a diversity problem. We've developed our ~~program~~ and lessons with diversity in mind. Engaging and supporting a diverse group of learners in technology is critical if we want to close the diversity gap.

TIPS FOR ENGAGING EVERYONE

Connect coding and technology to meaningful and creative projects.

Choose and/or adjust challenges so that they resonate with your learners' interests and the interests of their communities. Do they like music? Art? Animals? Food? Philanthropy? Is there an issue in their community that they have a tech-based solution for?

Connect coding and technology to meaningful and creative career paths.

There isn't just one technology job — there are thousands of jobs! Developers work on movies, games, medical devices, and more. Sharing these diverse and creative roles with learners can help broaden their understanding of the industry.

Maintain a social and collaborative learning environment.

Encourage group work, peer-to-peer mentoring, and recognition and demonstration of work and accomplishments. Facilitate ice-breaker activities and games to develop bonding among the participants, especially if they are not familiar with one another.

Recruit mentors who reflect the community.

When recruiting volunteers or inviting guests to be industry speakers, it is ideal to have them reflect the community. Learners will more likely connect with those who they can relate to. Diversity in mentors is important, and also highlights those who may be underrepresented in the industry.

Focus on why, not just how.

Loop lessons back to why coding matters — not just how to code. Find ways to link learning to your learners' diverse realities beyond the classroom. Girls, in particular, really resonate with how what they are working on can have an impact on and change the world.

Be aware of unconscious bias.

Keeping biases in check is important. We often unintentionally guide boys toward 'boy' things and girls toward 'girl' things. There are subtle biases in society that affect students like 'girls aren't good at math.' Keep in mind that you may have learners that hold these misconceptions. Take the time to debunk them if you identify them.

Be aware of imposter syndrome.

Imposter syndrome is strong for groups underrepresented in technology, and is often associated with high-achieving learners. Youth experiencing imposter syndrome might demonstrate feeling that they aren't "smart enough, good enough, or doing enough to succeed". Be aware of this with the youth you are working with, and if you identify it, make an effort to provide extra support, and reinforce these learners' skills and aptitudes, especially for subjects like math, science, and technology.

Learning Objectives

Coding is really about learning to solve problems, rather than learning a specific language or tool like Java or Scratch. Programming languages evolve and change all of the time, but the fundamentals of how you approach problem solving with computers doesn't change.

Computational thinking, or process for solving problems, can be taught even without learning a specific programming language. The code is just the tool that facilitates solving a challenge in a particular way. This is important to remember as educators! You don't need to know all the syntax of a language to teach it; you just need to understand the logic of solving problems. This logic has likely already been developed during your journey as an educator. Now, it's time to apply that logic to teaching code.

By teaching code, we are teaching:

Computational Thinking

- Logical reasoning
- Critical thinking
- Pattern recognition
- Solving complex problems by breaking them down into simpler parts
- Debugging problems
- Developing ideas from initial concepts to a final project

Concept About Computers

- Computer programs are created by humans and they tell the computer exactly what to do
- Computers aren't that bright or intuitive — they don't understand things the way humans do. You need to be exact and precise with your instructions to computers and instruct them step-by-step
- You don't have to be an expert to write code — you just need clear and careful thinking

Digital Citizenship

- Establishing a positive attitude towards building, not just consuming, technology
- Empowering kids to 'look under the hood' and ask questions about the technology they consume

Creativity and Collaboration

- Design Thinking
- Innovation

Programming Concepts

There are hundreds of computer programming languages out there and although they may look nothing alike to the human eye, at their core, they are all the same. There are fundamental concepts and ways to interact with a computer. We'll be using Scratch, Mozilla Thimble, and other tools to help us teach kids these concepts in a fun, relevant way.

- Variables
- Data
- Events
- Sequencing
- Iteration and loops
- Conditional statements
- Functions
- Parallel execution
- Boolean logic
- Random numbers
- Debugging
- Integrated math topics

CANADIAN HISTORY AND CULTURE

We've built lessons that will touch on various aspects of our history. Participants will learn about important historical events, and stories and experience them in a whole new way. We've built these lessons as ~~complements~~ to existing school curriculum, enabling educators to meet traditional learning objectives code.

TAKING A DEEPER LOOK AT

Computational Thinking

Computational thinking involves identifying a problem and articulating the solutions in a way that a computer (or another human) could act on. It involves breaking big problems into smaller parts, and describing specific steps to overcome these smaller challenges. Computational thinking concepts aren't unique to coding — you'll notice they are the principles many people use in their day-to-day lives to solve problems of all sorts.

Computational thinking involves the following key steps:

✓ **Logical Reasoning**

Computers aren't intuitive — they are predictable. Logical reasoning allows kids to work out what a computer will do. For example, if they hit the red 'X' button on their browser, they will close the current window. This process comes very quickly to kids as they develop a mental model for what technology does early on.

✓ **Algorithms**

Algorithms are a set of instructions that direct a computer to complete some task. Algorithms are common in our everyday lives - a lesson plan is an algorithm for a class, while a recipe is an algorithm for making our favourite dish. Writing out step-by-step instructions in plain English is what we call 'pseudo code'.

✓ **Decomposition**

Code is complex and problems are complex. An important part of computational thinking and coding is breaking down problems into smaller, more manageable, steps like we might break down a book report into different sections.

✓ **Abstraction**

After we break problems into smaller parts, abstraction helps us decide what's important and what's not. It helps manage complexity, like when we decide what information is needed to help solve a math equation or word problem.

✓ **Patterns and generalizations**

Patterning is an important part of learning to code. It helps us make predictions, create rules and solve more general problems. For example, a formula in math is a generalization that helps us solve many different problems.

GLOSSARY OF KEY

Programming Concepts

Here are some key programming concepts to help you understand programming better. You'll notice many of these concepts come up in other subjects and daily life, and aren't as daunting as you might think!

Algorithm: a step-by-step set of operations to be performed to help solve a problem

Array: a special variable that can store more than one value at a time; items are ordered by a number so that we can access them later (ie. an array called 'dogs' might have items: chihuahua, pug, and retriever)

Boolean Logic: 'and', 'or', 'not' are examples of boolean operators; the values you are working with must be either true or false (ie. if I am warm AND dry, then I am comfortable)

Conditionals: making decisions based on conditions (ie. if it is raining, then open your umbrella)

Debugging: finding problems in code and solving them

Events: one thing causing another thing to happen (ie. 'when green flag is clicked' block in Scratch)

Function: a type of procedure or routine that performs a distinct operation; there are often canned functions that exist already like the 'If on edge, bounce' block in Scratch

Loops: running the same sequence multiple times
(ie. 'repeat' or 'forever' blocks in Scratch)

Modularizing: exploring connections between the whole and the parts

Operators: mathematical and logical expressions (ie. 'X + X' block in Scratch)

Parallelism: making things happen at the same time

Remixing: taking an existing project or idea and making it new by changing or adding to it

Sequence: identifying a series of steps necessary to complete a task; computers read and perform commands in order from top to bottom

State: 'state' in a programming sense is just the same as 'state' in a non-programming sense (ie. the TV is on or off). Variables have states, values don't. For example, 42 is 42 and there's nothing you can change about it.

Syntax: the spelling or grammar of a programming language; the blockly structure in languages like Scratch removes the need for syntax

Variable: stores a piece of information that changes over time (ie. the score variable in a game may record the number of points a player has at any given time)

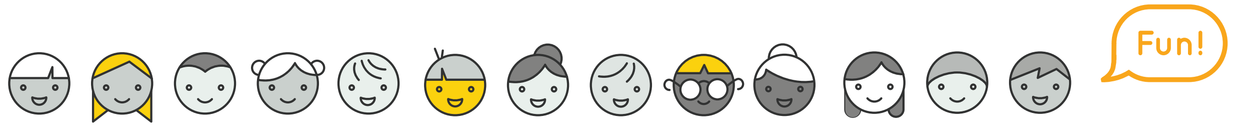
Lesson Planning

There are many ways to bring coding to your youth! You could start an after-school coding club or incorporate coding into pre-existing classes. For example, instead of having your students write a report on a historical figure or event, you might ask them to create a website instead! The possibilities are endless!

Minimum Requirements

YOU WILL NEED:

- ☐ **An Educator**
  To supervise students and facilitate sessions and coding challenges.
- ☐ **Volunteer Mentors**
  To support learners with their questions as the educators lead the session. Consider asking more advanced students to play this role. Optional, but highly recommended.
- ☐ **Learners**
  We've included a poster in the 'Printables' section you can use to help promote.
- ☐ **Venue**
  Somewhere to meet!
- ☐ **Access to Hardware**
  Laptop or desktop computers (at least 1 per pair of learners)
- ☐ **Extension cords and power bars**
  Projector, projector screen, and appropriate dongles to connect (or another way to display your screen to learners)
- ☐ **Access to Internet**
  WiFi connection for all laptops and desktop computers being used. This is optional. Some tools (ie. Scratch) can be used without an internet connection, while it is a strict requirement for others (ie. Mozilla X-Ray Goggles).



Preparation Timeline

WHEN	TASK
1 Week To Go	Go to **teacherslearningcode.com** and: 1. Read the Lesson Plan (Description to Extensions sections) 2. View the Example Project 3. Use the Solution Sheet to practice going through the activity 4. Review any videos and additional background information provided to prepare for the topic or theme of the lesson (if applicable)
3 Days To Go	Create any necessary online accounts (if applicable)
1 Day To Go	Print a copy of the Lesson Plan and Solution Sheet to have on hand in case the WiFi goes down Ensure you have all necessary hardware and supplies
Day-of	Set up hardware needed for the session Write any login information on chart paper/whiteboard/chalkboard, so it is easily read by all students

Setting the Tone

Before diving into a lesson, it's important to set the tone for learners by going over a few key concepts and values to guide their learning. We like to encourage and instill a growth mindset or 'fail-forward' approach in learners. Acknowledging upfront that technology fails and that's part of the learning process is critical in setting this tone. Encourage exploration, trial and error and collaborative problem solving. And among all else? Patience. Learning to code is like learning a new language and that takes time, patience, and lots of practice.

Here are some common things we like to address in all of our programs:

1. Technology Fails

If a computer isn't working the way it should and a learner is feeling frustrated, acknowledge that this is common. Try to troubleshoot the problem together. Ask what they would do if this happened at home. Would they quit and reopen the program? Restart the computer? Check if the computer is plugged in? If the problem does not resolve itself, try using Google as a resource! It is important to show the learner how to be resourceful for the future.

2. Learner-Driven Problem Solving

Encourage your students to ask others for help first before coming to you by using the 'Ask three, then me' protocol. Often other learners can troubleshoot many tech challenges and it's a great opportunity to empower them as leaders. You can also consider creating a troubleshooting checklist with your students. And when all else fails? Google is your friend! Resourcefulness and learning to access information on the Internet is an extremely valuable skill for everyone to learn and is strongly encouraged.

3. Inquiry-Based Learning

We've developed many challenges to introduce fundamental programming concepts to youth, but we stress that these are just starting points. Allowing students to run with their ideas and questions is an important part of the learning experience.

SUGGESTED STRUCTURE

Of A Lesson

Each lesson will follow the same format:

- **Introduction**
- **Code-Along**
- **Activity**

The **Introduction** sets the stage for the lesson by providing some background on the theme in focus. This is where you should share our 'Setting the Tone' tips.

Next up, during the **Code-Along**, you will introduce the tool being used (e.g. Scratch or Thimble). Give learners a chance to play around with and test out the tool, with a bit of structure created through the 'challenges' provided. It is here that you will set the expectation of using problem-solving and resourcefulness to figure out how to make things happen with code. Do this through asking (rather than telling) learners where to find things, verbalizing your thought process from A-to-B, and constantly redirecting questions towards other learners, or towards the reference.

The third part of the lesson, which consumes the most time is the **Activity**. While the beginning may look similar to the Code-Along portion as learners get acquainted with the provided activity, the majority of the time will be spent brainstorming remix ideas to personalize their project. While learners remix, be sure to check-in often and have volunteers come up to the front of the class to share cool findings and solutions with the group.

Finally, the end of the Activity portion should include demonstrations, where possible! We recommend having learners come up one at a time to present their projects. However, if you're short on time, you can also have a Gallery Walk, where participants open their laptops, tuck in their chairs, and walk around the room to explore each other's final projects.

Feel free to include breaks whenever a group needs a stretch break or a change of pace.

Pacing

One of the most important skills needed for teaching with technology is flexibility. Things happen, tech fails, and your learners will all come from a variety of skill levels and backgrounds.

A large part of delivering a successful coding lesson is your ability to read the group. Your goal is to teach at the level of the majority of your learners - and to try to keep everyone engaged, whether that means offering extra challenges to learners that are ahead, or modifying activities for those who are falling behind.

Each lesson presented will have a Solution Sheet with the steps required to create a basic working project. They also include a number of add-ons or challenges for learners that are ahead of the group. You can support advanced learners by giving them challenges from this list, or providing them with a copy of the solution sheet to work from. Please also ~~used~~ the 'Extensions' section of the Lesson Plan to support learners who are moving ahead.

In order to avoid leaving learners behind, try not to assume anything. Check in with the group and ask questions like: "What is.. (e.g. a browser)?" or "What does.. (e.g. interactive) mean?" Also, ask learners for reminders on things like keyboard shortcuts, and repeat information as much as needed in order to help it really stick.

Assess the group by looking at facial expressions or asking learners to give a thumbs up or a thumbs down for feedback on how they are doing. Try to ask the right kinds of questions. Instead of asking "Does everyone understand? Are we good to move on?" say "If you need one more minute, raise your hand," or "If your screen does not look like mine, raise your hand." The more specific, the better.

Ideally, each lesson will end with learners having completed a personalized project, and having had the opportunity to present to the group. However, if this isn't possible, for whatever reason, and you aren't able to get through everything... that's okay! Ultimately, we want learners to have a positive experience with technology, and have fun!

CONTINGENCY

Planning

The reality is that technology fails — the internet goes down, computers crash, work is lost — and this often feels like failure, which is not a feeling any teacher or student likes.

Below we've prepared some helpful tips for you to mitigate and be prepared for technology issues:

PLAN AHEAD

The more you can plan ahead for potential tech hiccups, the more confident you will be when class starts.

Try these suggestions:

- ☐ Set up computers ahead of time
- ☐ Write logins and passwords on the board
- ☐ Use one universal login versus one per student
- ☐ Charge laptops before class
- ☐ Encourage students to partner and work together

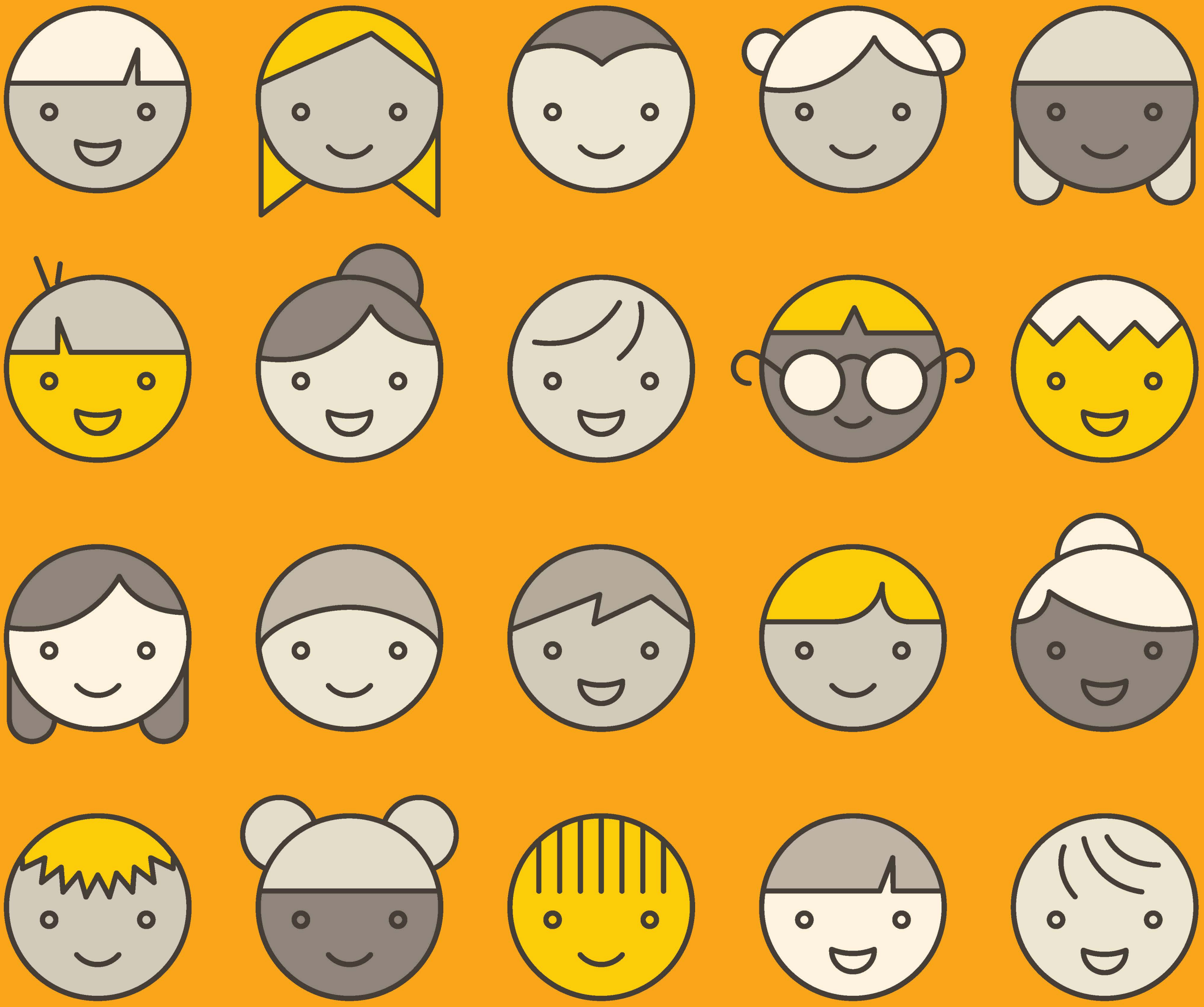
PAPER-BASED OR UNPLUGGED ACTIVITIES

You don't need the internet or even a computer to teach coding or computational thinking! There are a lot of ways to teach fundamental programming concepts using unplugged activities. See 'Teachers Learning Code Digital Toolbox: Unplugged Activities' for more details.

DATE:

DATE:

DATE:



GET IN TOUCH

129 Spadina Ave., Suite 501
Toronto, ON M5V 2L3

info@canadalearningcode.ca
canadalearningcode.ca

※teachers learning code