Robotics in the Classroom

Competency

Educator writes and teaches code using robots.

Key Method

Educator uses a range of robots and pedagogical strategies to teach students to code.

Method Components

What is coding?

Stated simply, coding is “the language that computers speak.” Much like “human languages,” many different coding languages are used around the world, all of which have different strengths and uses.

To use computers as a tool to shape the world around us, we need to learn these languages. Luckily, coding languages are based on a system of rules; when we write code, we provide a set of instructions for a computer to follow in a way that it understands. Once we understand the fundamental set of rules that underpin coding as a whole, we can start to apply those rules to all the different coding languages and their unique syntax.

Why should students learn to code?

One of the reasons most often shared for why students should learn to code is “to prepare them for the jobs of the future.” This is a worthwhile goal; jobs in STEM fields are growing at a rate of close to 8%, compared to just 3.7% for non-STEM jobs, and over 70% of jobs in STEM are computing jobs or use computer science in a major way (US Bureau of Labor Statistics, 2021). Coding can be a valuable skill in the job market.

However, many students will not end up working in a STEM field. Nevertheless, these students should still learn how to code!

Coding is not only an enjoyable pursuit for many, but also, one of the goals of 21st-century education should be to move students from mere digital consumers to digital creators. Rather than being passive users of technology, students should have a deeper understanding of how technology is shaping their lives and societies and how they, in turn, can use technology to make a positive impact in the world. To truly understand how technology works, you have to understand the fundamentals of code and computer science. When students learn to code, what they are really learning is how computers work.

Furthermore, coding is an important and effective way for students to learn key 21st-century skills such as communication, collaboration, critical thinking, and creative problem solving.

Why use robots to teach code?

Coding and robotics are often used to mean the same thing in the K-12 education sphere, but while they are related they are actually separate learning experiences. Coding can be taught separately from robotics (and you are encouraged to do so). However, teaching coding through the specific context of programmable robots can be an excellent tool for engaging students in powerful learning experiences.

Learning to code with robots not only allows students to explore a valuable real-world application of coding, but it is also extremely hands-on. Rather than merely watching the outcome of their code on a screen, the specific application of robotics allows students to see the outcome of their code in the "real world" and get a hands-on perspective on how the code they create influences, and is influenced by, the world around them. For example, how might the unevenness of a flooring surface influence the code they write? How might the angle at which a robot brushes up against a wall influence its path of movement? These kinds of practical problems are highly motivating for students and give them a perspective on coding that they would not receive from purely computer-based experiences.

Building vs Programming Robots

When first hearing about robotics, many students (and teachers!) can become overwhelmed by the thought of having to build their own robot. While learning about robot design and engineering is absolutely valuable learning, and some K-12 robotics tools such as the LEGO Ev3, LEGO Spike, and mBot do incorporate some level of robot construction, the focus in the traditional K-12 sphere is primarily on learning to code pre-built robots.

In this micro-credential, we will focus exclusively on programming/coding through the application of robotics; robot design and engineering will not be discussed.

Foundations of effective coding instruction

Though introducing STEM (and coding, more specifically) in the classroom has increasingly become a stated priority of schools and governments across the globe over the last decade,

understanding how to effectively integrate coding in the classroom at the elementary level remains inconsistent.

According to research by the Computer Science Teachers Association and other organizations that develop best practices, the five foundations of effective coding instruction are:

1. **Demonstrating knowledge and skills thorough computer science (CS)**

No, you do not need to be a coding expert to integrate coding in the classroom. However, educators do need to take the time to experience coding themselves and build their skills to effectively teach coding to elementary students.

Educators should have a solid understanding of fundamental coding concepts (e.g., sequence, conditionals, loops, functions, etc.), have basic familiarity with at least one programming language (e.g., a block-based language like Scratch or a text-based language like Python), and be able to recognize common student misconceptions and areas of difficulty.

By taking the time to familiarize themselves with the foundations of coding, educators will be both better equipped and more confident in their ability to bring high-quality coding instruction into their classroom.

2. **Implementing evidence-based practices as responsive classroom practitioners**

Every classroom is unique, but there are general principles and evidence-based practices that should guide all educators in bringing high-quality coding instruction to their students, including:

1. *Hands-on learning.* While fundamental theoretical concepts underpin all programming languages, students need hands-on experience with coding to truly understand and develop the skills they need.
2. *Reading, writing, and altering code.* Learning to code is about more than just writing the code yourself. Students should also be given opportunities to “read” code (i.e., look at prewritten code and accurately describe what the program will do) and to alter code (i.e. be given prewritten code and describe the effects of changes).
3. *Troubleshooting.* When it comes to code, it is not a matter of *if* you will make a mistake, but *when*. On top of learning the fundamental principles of coding and getting hands-on experience writing code, students also need explicit instruction and practice with troubleshooting strategies.
4. *Multiple pathways.* There is almost always more than one way to achieve a given outcome when programming. Effective educators of programming account for this inherent flexibility when assessing student work while still encouraging students toward efficiency and simplicity in their work.
5. *Process vs. product.* Though the ultimate goal is to have a working project, effective computer science educators recognize that it is the process of coding, troubleshooting, and refining projects, rather than the final project itself, that carries the greatest weight in assessing and evaluating student learning.

3. **Designing learning experiences using pedagogical content knowledge**

Though many pre-made lessons and activities exist to support educators in bringing coding instruction to their classroom, educators must be able to evaluate and modify these resources as well as create their own original learning experiences to best meet the needs of their unique group of students.

4. **Continuously developing knowledge, practice, and professional identity**

An "hour of code" is a great place to start, but not to end. For an educator beginning their journey with coding in the classroom, it can be tempting to stay within their comfort zone with simple coding lessons and activities they are familiar with. However, students may quickly advance past these learning experiences and require more advanced and novel experiences.

Therefore, effective coding instruction means that learning is never done! Educators should be continuously evolving their own skills and looking for ways to engage students in increasingly complex learning.

5. **Advocating for equity and inclusion in the CS classroom**

Educators must recognize that the world of STEM, and computer science more specifically, has not been equitable in the past or present. Women/girls, racialized people, people with disabilities, and other marginalized groups remain deeply underrepresented in the world of computer science.

We must address these issues at an early age. For girls, research demonstrates that by age 12, they have begun to like math and science less, expect not to do as well in these subjects, and attribute their failures to lack of ability (Sawyer, 2016).

These problems will not fix themselves. Cultural biases about who belongs in the world of STEM and computer science exist in us all, even at a subconscious level. Beyond merely teaching coding, educators must take an active role in promoting equity and inclusion in their coding instruction and overall classroom environment for all students to succeed and recognize their potential.

Supporting Rationale and Research

Dweck, Julia. (Jan. 29, 2016). *Unlocking the code for robotics in the classroom.* https://www.edutopia.org/blog/unlocking-code-robotics-in-classroom-julia-dweck

Geist, Eugene. (2016). Robots, programming and coding, oh my*! Childhood Education, 92*(4), 298–304. https://www.tandfonline.com/doi/abs/10.1080/00094056.2016.1208008

Johal, W., Castellano, G., Tanaka, F., et al. (2018). Robots for learning. *Int J of Soc Robotics 10*, 293–294 https://doi.org/10.1007/s12369-018-0481-8

Catlin, D. (2019). Beyond coding: Back to the future with education robots. In L. Daniela (ed.) *Smart learning with educational robotics*. Springer, Cham. https://doi.org/10.1007/978-3-030-19913-5_1

Canada Learning Code. (n.d.). Learning for the digital world: A pan-Canadian K-12 Computer science education framework. https://k12csframework.ca/wp-content/uploads/Learning-for-the-Digital-Future_Framework_Final.pdf

Canada Learning Code. (n.d.). Teachers learning code digital toolbox: Quick start guide. https://www.canadalearningcode.ca/wp-content/uploads/teacher-quick-start-guide.pdf

Reach. (2017, August). Field report on K–12 computer science. https://drive.google.com/file/d/0B2eCjHNmaBGZeGpwTGlRTUJKZlU/view

Computing at School. (n.d.) Computational thinking: How do we think about computing so that computers can help? https://community.computingatschool.org.uk/files/8221/original.pdf

International Society for Technology in Education (ISTE), Computer Science Teachers Association (CSTA). (2011). *Computational thinking teacher resources* (2nd ed.) https://cdn.iste.org/www-root/2020-10/ISTE_CT_Teacher_Resources_2ed.pdf?_ga=2.206381526.472047317.1620655308-1370069785.1620246492

Computing at School. (n.d.). What is programming? https://community.computingatschool.org.uk/files/8222/original.pdf

International Society for Technology in Education (ISTE). (n.d.). Computational thinking competencies. https://www.iste.org/standards/computational-thinking

Computer Science Teachers Association (CSTA). (2012). Special issue computer science K-8: Building a strong foundation. https://csteachers.org/documents/en-us/25b0b43f-0817-4c08-9210-f1a96f2b1c7d/1/

Goode, D., et al. (2020, January 21). Computer science for equity: Teacher education, agency, and statewide reform. https://csteachers.org/documents/en-us/8fe1ba43-45fd-4360-8820-2e0dd33a1252/1/

Google. (2014, May 26). Women who choose computer science: What really matters. https://csteachers.org/documents/en-us/c0015462-38a7-41e2-b9fb-b48107953e28/1/

Sawyer, E. (2016, January 5). *Why STEM's Future Rests in the Hands of 12-Year-Old Girls*. TechCrunch. https://techcrunch.com/2016/01/05/why-stems-future-rests-in-the-hands-of-12-year-old-girls/

Smith, L., et al. (2017, September). Cracking the code: the prevalence and nature of computer science depictions in media. https://csteachers.org/documents/en-us/db6c4cf7-62d6-4d0f-921c-658a540cc10f/1/

Zilberman, A., & Ice, L. (2021, January). Why computer occupations are behind strong STEM employment growth in the 2019–29 decade. *Beyond the Numbers: Employment & Unemployment*, *10*(1). (U.S. Bureau of Labor Statistics). https://www.bls.gov/opub/btn/volume-10/why-computer-occupations-are-behind-strong-stem-employment-growth.htm

Resources

Standards
Computer Science Teachers Association K-12 Standards
https://www.csteachers.org/page/standards

Lesson Plan Template
https://dl.dropbox.com/s/gkw7sn841l5kb76/Lesson%20Plan%20Template.docx?dl=0

Terms and Concepts

- Algorithm: a single set of step-by-step instructions that tells a computer how to perform a task.
- Program: a set of step-by-step instructions that tells a computer how to perform a task. A program can be simple (including just one algorithm) or complex (including multiple algorithms).
- Sequence: the order in which your computer will run your code. A computer will run your code in the exact order it is written in, so the order in which you write your code matters.
- Event: an event is something a computer is always looking for; e.g., the user pressing a button. Events are often used to trigger other actions within a program; e.g., when the user presses the A key (event), the symbol A will appear on the screen (action).
- Loop: a repeating section of code. A loop will repeat until a certain condition is met; e.g., repeat # of times, repeat until something else happens, repeat infinitely, etc.
- Conditional: an “if...then…” statement. The computer will choose between a set of options based on whether a condition is true or false.
- Variable: can store data. It is associated with a symbol or name that can be referred to throughout the program.
- Function: a “sub-program” within code that performs a specific action. A function is often used to store a series of actions that will be required multiple times throughout the program but not necessarily one after the other. By assigning a name to the function, it can be “called” at any point in the program to run the series of actions.
- Programmable robot: a pre-built robot that allows users to write code to control the actions and movements of the robot.

Submission Guidelines & Evaluation Criteria

To earn the micro-credential, you must receive a passing evaluation for Parts 1 and 3, and a "Yes" for Part 2.

Part 1. Overview Questions

Please write your responses below (1000-word limit for the five questions in total).

1. Describe your role in education. What grade and subject or content area do you teach? What should we know about you and your classroom?
2. What is your current level of experience and confidence with teaching coding/robotics?
3. Identify at least one asset and one barrier you anticipate to integrating coding/robotics into your classroom.
4. Why do you believe it is important to teach students to code?
5. What are you hoping to gain through this micro-credential?

Passing: Response provides reasonable and accurate information that outlines the prior experience of the educator and the context of their classroom/teaching. Educator specifies a learning goal that describes what they hope to gain from this experience. Educator outlines their current mindset and experience when it comes to teaching coding in sufficient detail.

Part 2. Work Examples/Artifacts/Evidence

To earn this micro-credential, submit the following three artifacts:

Artifact 1: Certificates of Completion

1. Certificate of completion for a STEM Minds Teacher Professional Development Workshop related to your STEAM Hub course (see Artifact 2 below)
2. Certificate of completion for ONE of the following STEAM Hub courses:
    a. Robotics with Dash
    b. Robotics with Sphero
    c. Robotics with LEGO Ev3

Artifact 2: STEAM Hub Course Final Project

For the STEAM Hub course you selected above, please submit a copy of your final project. It must include the following:

- The full project file; please do not submit screenshots
- Any relevant share settings appropriately set to allow anyone to view the project

Artifact 3: Lesson Plan

Submit a lesson plan showing how you will bring this coding experience to your classroom. This lesson may be a “stand alone” lesson or one in a larger unit. Please indicate this context for the lesson somewhere in the lesson plan. You may use your own lesson plan template or the suggested template in the Resources section. Your lesson plan must include the following information:

1. What core concepts you plan to introduce to students and how you plan to do so
2. What project(s) students will be asked to create and how they will have the opportunity to test and refine them
3. How you intend to foster an inclusive and collaborative coding culture in your classroom, with a focus on historically underrepresented groups instead (including girls/women, students with disabilities, ELL students, etc.)
4. How you plan to address common student misconceptions/areas of difficulty
5. What troubleshooting strategies you intend to introduce to students. Please also include *how* and *when* you plan to introduce these strategies.
6. What opportunities students will have to communicate about coding
7. How you intend to assess and evaluate student work with a focus on process over product

Part 2. Scoring Guide

Artifact	“Yes”	“Almost”	“Not Yet”
Artifact 1	The certificate of completion for both the course and the professional development workshop were provided.	N/A	One or both of the certificates are missing.
Artifact 2	The project provided meets the expectations as outlined in the project rubric within the STEAM Hub course at a level of 80% or higher.	The project provided meets the expectations as outlined in the project rubric within the STEAM Hub course at a level of less than 80%.	The project was not provided.
Artifact 3	The lesson plan includes all of the following: 1. Core concepts to be addressed 2. Project description 3. Inclusion and collaboration strategies 4. Anticipated student misconceptions/ areas of difficulty 5. Troubleshooting strategies to be taught	The lesson plan includes some of the following: 1. Core concepts to be addressed 2. Project description 3. Inclusion and collaboration strategies 4. Anticipated student misconceptions/ areas of difficulty 5. Troubleshooting strategies to be taught 6. Opportunities for student communication	The lesson plan includes only one or two of the following: 1. Core concepts to be addressed 2. Project description 3. Inclusion and collaboration strategies 4. Anticipated student misconceptions/ areas of difficulty 5. Troubleshooting strategies to be taught 6. Opportunities for student communication

	6. Opportunities for student communication 7. Assessment and evaluation plan	7. Assessment and evaluation plan	7. Assessment and evaluation plan

Part 3. Reflection

Please write your responses below (1000-word limit for the five questions in total).

1. Throughout this experience, what steps did you take to foster an inclusive and collaborative coding culture in your classroom? What impacts did these have on you and your students?
2. How did this micro-credential process influence how you teach coding/robotics?
3. What were the shared challenges your students faced in learning coding with robotics? How did you address these challenges?
4. In what ways did your students engage with collaboration, communication, critical thinking, and creative problem solving through this experience?
5. How would you describe your students' overall experience with coding/robotics? If you had to do it again, what would you do differently? What would you do the same?
6. What are your next steps for growth as an educator in this area?

Passing: Response provides reasonable and accurate information that outlines an approach to inclusivity in teaching coding/robotics. Educator explores how the experience influenced their teaching and their next steps for growth. The response outlines the impact on the students and their experience in sufficient detail.