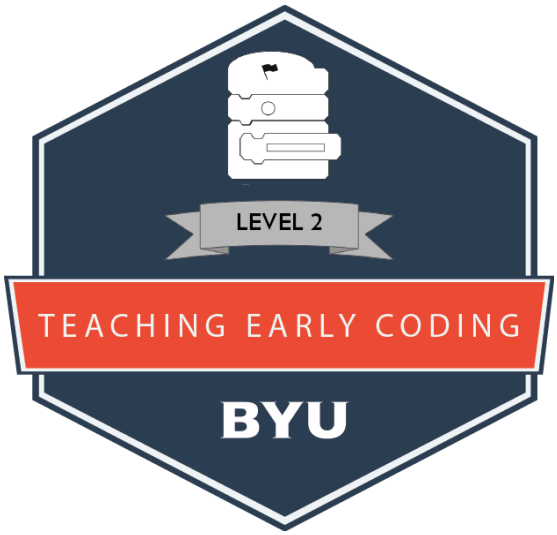


Teaching Early Coding - Level 2

Badge Guide

Enoch Hunsaker
Brigham Young University
2018

Table of Contents

Badge Introduction

Coding is an increasingly important subject to consider teaching to young learners in the 21st Century. In fact, many consider coding to be a literacy of the 21st Century that needs to be learned just like reading and writing in a native language (Code.org, 2013).

In the process of earning this badge, you will learn the following coding concepts:

- While Loops
- For Loops
- Variables
- Functions
- Parameters
- Operators

You will also learn how to find resources and create activity plans to teach these coding concepts in your classroom and integrate them with other academic subjects.

It is estimated that this badge should take you approximately **2.5 hours** to complete.

Recommended Prior Knowledge

This badge is intended to be completed **after** the Teaching Early Coding Level 1 (http://bit.ly/2wlQjFN) badge. Additionally, some requirements of this badge will draw on the assumption that you understand the fundamentals of computational thinking (CT). If you aren't familiar with CT, you may consider first earning BYU's Understanding Computational Thinking (http://bit.ly/2wdA11g) badge.

International Standards Alignment

ISTE Standards for Educators

- **Learner Standard 1a:** Set professional learning goals to explore and apply pedagogical approaches made possible by technology and reflect on their effectiveness.

- **Facilitator Standard 6c:** Create learning opportunities that challenge students to use a design process and computational thinking to innovate and solve problems.

ISTE Standards for Students

In particular, this badge helps meet the ISTE Facilitator Standard (Standard 6) by empowering teachers to facilitate the following standards for students:

- **Empowered Learner Standard 1d:** Students understand the fundamental concepts of technology operations, demonstrate the ability to choose, use and troubleshoot current technologies and are able to transfer their knowledge to explore emerging technologies.
- **Computational Thinking Standard 5:** Students develop and employ strategies for understanding and solving problems in ways that leverage the power of technological methods to develop and test solutions.

Intended Learning Outcomes

1. Apply advanced coding concepts in a variety of contexts.
2. Facilitate student learning of coding concepts in effective and developmentally-appropriate ways.

Requirement 1.1: Coding Concepts

Note: Teaching Domain *in this badge refers to the subject, grade level, and demographic in which you teach or plan to teach.*

Choose a block-based programming platform (e.g., code.org Express, Scratch) and build one or more block-based code sequences to show that you can apply ALL of the following coding principles (you may build these sequences either as part of an activity, or as a stand-alone project):

- *While Loops*
- *For Loops*
- *Variables*
- *Functions*
- *Parameters*
- *Operators*

Take one or more screenshots of your sequence(s) and include them on your submission form along with annotations that describe which coding principles they demonstrate.

BYU's Teaching Early Coding badge levels (level 1 and level 2) were designed on the sequencing found in Code.org's CS Fundamentals Express course (https://studio.code.org/s/express). The most direct way to learn the six coding concepts necessary for this badge is to browse through and complete several activities in the second half (lessons 17-30) of that course. You need not necessarily complete every single item within these lessons. Rather, work at a concept until you grasp it, then take a screenshot, add it to your submission form, and annotate it. One screenshot may include multiple or even all of the coding concepts as desired. If you are a little more familiar with the concepts, or if you prefer, you may also choose to use Scratch (https://scratch.mit.edu/), a block-based programming language developed by MIT that is similar to Code.org, but which will not provide you with as much guidance throughout the process.

You may want to reference the information below before, during, or after your block-based coding practice to solidify your understanding of these concepts.

While Loops vs. For Loops

As you should have already learned in the first badge of this sequence, a loop is a programming structure that repeats a command (or set of commands) a specified number of times or until a certain condition is met. *While Loops* and *For Loops* are simply two forms that loops can take.

A while loop allows code to be executed repeatedly based on a condition. When a while loop is run, the code is executed *while* a certain condition is true. If the specified condition is never false, the commands within the loop will run indefinitely—or crash your program. A for loop, on the other hand, will repeat only a specified number of times.

To better understand this concept, take a look at the two clips below:

Video 1. Differences between while & for loops (in Python). Time: 0:00-1:46. Click, copy, or type this link into your browser to view: http://bit.ly/2JTQrhg. ***Note:*** *This video is copyrighted by Barron Stone under a Standard YouTube Licence and is therefore not included in the CC-BY license of the rest of this document.*

Video 2. Differences between while & for loops (in Python). Time: 3:52-5:01. Click, copy, or type this link into your browser to view: http://bit.ly/2JSyYpv. ***Note:*** *This video is copyrighted by Barron Stone under a Standard YouTube Licence and is therefore not included in the CC-BY license of the rest of this document.*

In the block-based code that you and your students will most likely be using, while loops and for loops also look slightly different:

While Loop

For Loop

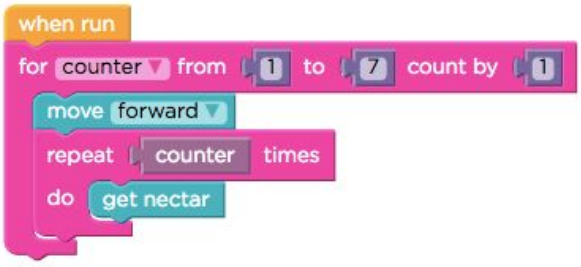


Image 1. While Loop vs. For Loop in Code.org. Note that each loop begins with the appropriate word ("while" or "for") and that the while loop specifies a condition to be met whereas the for loop specifies a starting ("from") point, an ending ("to") point, and a rule for incrementing the counter by 1 after each iteration.

Variables

A variable is a container for storing a value that can be defined, referenced, and updated as the program runs. Variables are widely useful in other programming structures. For example, the for-loop image above depicts the "counter," which is a key mechanism that

drives the for loop forward. That counter is actually a *variable* or a container for storing a value. The loop above created that variable by naming it "counter" and assigned it an initial value of "1." After that, the loop specified that the variable called "counter" should be incremented by 1 every time the loop ran, so the value within the container increased by 1 every time the loop completed a cycle.

A variable is highly useful for storing anything in the program that needs to change, such as a game score or anything entered by a user into an input field.

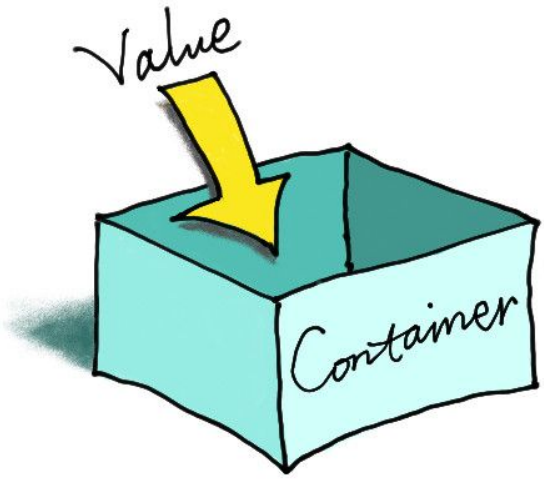


Image 2. A variable is a container that stores a value. Image retrieved 4 May 2018 from https://crumplab.github.io/programmingforpsych/fundamentals-of-computer-programming-languages.html

For a concise explanation of variables, watch this video:

Video 3. Olga from Amazon introduces variables. Click, copy, or type this link into your browser to view: http://bit.ly/2wd7MQF. ***Note:*** *This video by Code.org is released into the Creative Commons under a CC-BY license.*

For a more complex understanding of variables, consider watching this video as well:

Video 4. CS Principles: Intro to Variables - Part 1. Click, copy, or type this link into your browser to view: https://youtu.be/G41G_PEWFjE. ***Note:*** *This video is copyrighted by Code.org under a Standard YouTube Licence and is therefore not included in the CC-BY license of the rest of this document.*

Many of the examples above have referred to variables as holding numbers, but it is important to note that variables can store many types of things. The most important of these are

- numbers

- booleans (e.g., a true or false value)
- strings (e.g., any combination of characters, such as a word or password.)

Functions

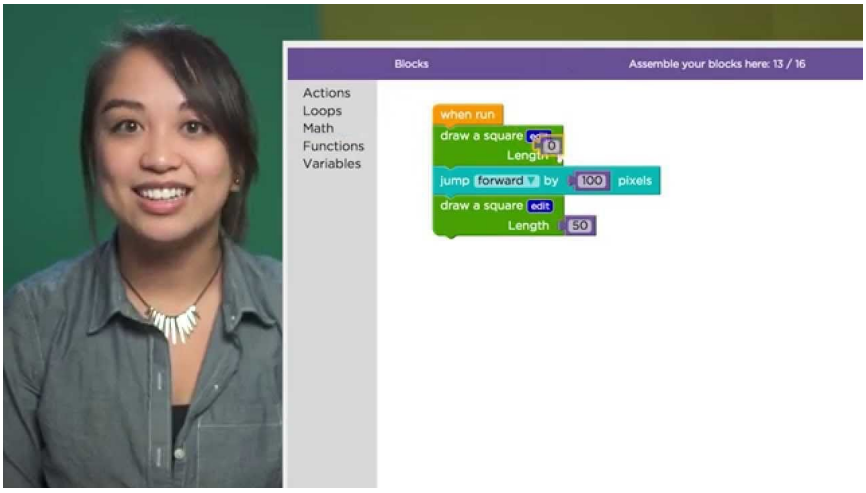
A function is a user-defined action that can consist of any number and combination of commands or other functions. It can be “declared” (defined) and “called” (used). Watch this video to better understand what a function is, how it works, and what it does. For bonus points, also pay attention to how the person in the video references computational thinking (CT) principles in her explanation!

Video 5. CS Principles: Defining and Calling Functions. Click, copy, or type this link into your browser to view: https://youtu.be/yPWQfa4CHbw. ***Note:*** *This video by Code.org is released into the Creative Commons under a CC-BY license.*

.

Parameters

Parameters are variables stored within a function that change how the function works. As you watch this video, consider how parameters can make functions almost infinitely more useful.



Video 6. Functions with Parameters. Click, copy, or type this link into your browser to view: https://youtu.be/e9qjXKaeDHg. ***Note:*** *This video by Code.org is released into the Creative Commons under a CC-BY license.*

Operators

Operators are characters that can be used to assign, compare, and change variables and parameters. There are several types of variables, and some of the most important are the following:

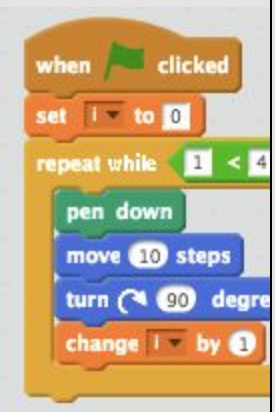
- Arithmetic (+, -, *, /) Comparison (<, >, <=, >=, ==, ===)
- Boolean (AND, OR, NOT)
- Assignment (=)

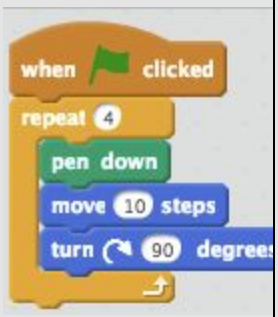
Operators can be used in conjunction with parameters in functions to make the function even more useful. For an example of this, see the table below. Note how the use of functions and operators combined can help you transform a function for drawing a square of fixed side length into a function that allows you to draw a regular polygon of any shape and size.

Comparison Table

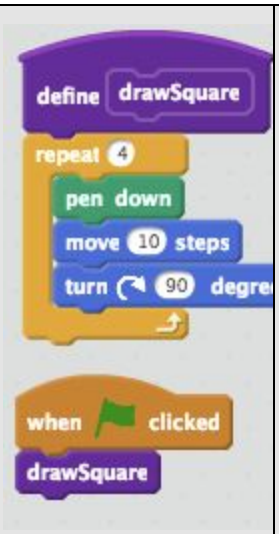
The following table may be useful to you as you consider each of the concepts introduced in this badge.

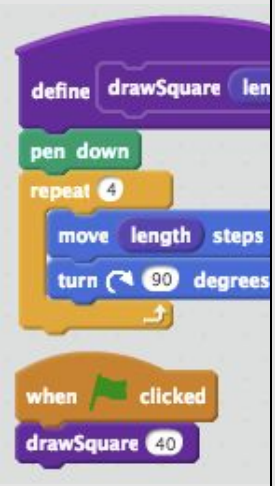
Coding Concepts: Level 2				
Concept & Definition	Example	Block-Based	Javascript	Output

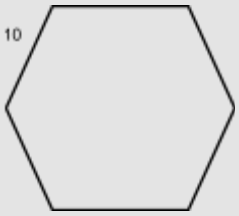
While Loops repeat a set of steps until a certain condition is met (can be infinite).	Repeat until the number of sides is 4:
	• Put pen down
	• Move 10cm
	• Turn 90° right.
			`var i=1;`
			`while (i<4) {`
			`  penDown;`
			`  move (10);`
			`  turn (90);`
			`  i=i+1;`
			`}`
				

For Loops have a pre-specified beginning, end, and increment.	Starting when i=0 and increasing i by 1 each iteration, repeat until i<4:
	• Put pen down
	• Move 10cm
	• Turn 90° right.
			`for (var i=1, i<4, i=i+1){`
			`  penDown;`
			`  move (10);`
			`  turn (90);`
			`}`
				

Variables are containers for storing a value that can be defined, referenced, and updated as the program runs.	Create a space for a score. Increase the score each time an enemy is destroyed.		`var score=0;`
			`enemy.onclick =`
			`disappear(){`
			`  enemy.style`
			`  .display =`
			`  "none";`
			`  score=score`
			`  +1;`
			`}`
				A score value that increases each time an enemy is destroyed.

Functions are user-defined actions that can consist of any number and combination of commands or other functions. It can be "declared" (defined) and "called" (used).	When I say "draw a square," do the following 4 times:
	• Put pen down
	• Move 10cm
	• Turn 90° right.
	
	Draw a square.
			`function drawSquare(){ for (var i=1, i<4, i=i+1){ penDown; move (10); turn (90); } }`
			
			`button.onclick {\ drawSquare(); }`
				

Parameters are variables stored within a function that change how the function works.	Design an algorithm that will easily create a square of any specified size.		`function drawSquare(length){ penDown; for (var i=1, i<4, i=i+1){ move (length); turn (90); } }` `button.onclick { drawSquare(40);}`	

Operators are characters that can be used to assign, compare, and change variables and parameters. • Arithmetic (+, -, *, /) Comparison (<, >, <=, >=, ==, ===) • Boolean (AND, OR, NOT) • Assignment (=)	Create an algorithm that can create a regular polygon of any shape and size.		`function drawPolygon(sides, length){ penDown; for (var i=1, i<sides, i=i+1){ move (length); turn (360/sides); }} button.onclick { drawPolygon (6, 10);}`	

Requirement 1.2: Learning Reflection

On your submission form, respond thoroughly to the following question:

List at least three (3) computational thinking (CT) skills, attitudes, or approaches that you feel were fostered by the activities you completed in Requirements 1.1. (If you don't know what computational thinking is, consider earning the Understanding Computational Thinking badge!)

Whether or not you have completed the Understanding Computational Thinking badge, you may find this table useful for reference. It outlines the basic computational thinking (CT) skills, attitudes, and approaches upon which you are expected to reflect in this requirement.

Components of Computational Thinking		
Skills	**Attitudes**	**Approaches**
• ***Decomposition***: Breaking down data, processes, or problems into smaller, manageable parts. • ***Pattern Recognition:*** Observing patterns, trends, and regularities in data. • ***Abstraction:*** Creating a visual model or simulation of a problem that incorporates only the most important details. • ***Algorithm Design:*** Developing the step by step instructions for solving this and similar problems. • ***Evaluation:*** Ensuring that your solution is a good one.	• ***Confident:*** Believing in one's own ability to solve problems. • ***Communicative:*** Willing and able to communicate effectively with others. • ***Flexible:*** Able to deal with change and open-ended problems.	• ***Tinkering:*** Experimenting and playing. • ***Creating:*** Designing and making. • ***Debugging:*** Finding and fixing errors. • ***Persevering:*** Keeping going. • **Collaborating:** Working together.

Table 2. Components of Computational Thinking

Requirement 2.1: Unplugged Activity

Identify (or create) an unplugged activity that could effectively teach two or more of the coding concepts listed in requirement 1.1 (i.e., while loops, for loops, variables, functions, parameters, and operators) to students within your teaching domain. On your submission form, include the following:

- *A link to (or description of) the activity*
- *An indication of which coding concepts it teaches*
- *A written justification of why this activity is appropriate within your teaching domain.*

Terminology

Unplugged Activity: An activity in which students use non-digital physical objects (e.g., a pencil and paper, games, manipulatives, or their bodies) to learn coding concepts.

Teaching Domain: The subject area, grade level, and student demographics of your classroom.

Unplugged Resources

A wide variety of instructional resources for unplugged activities are available for free online. These resources may include include lesson plans, printable worksheets, cutouts, etc., and sometimes even videos that depict an example lesson. Among the sites that make these resources available are the following:

- CS Unplugged (https://csunplugged.org)
- CS Fundamentals Unplugged on Code.org (https://code.org/curriculum/unplugged)

You can also find many resources simply by performing a Google search for “unplugged coding activities.”

In addition to these online resources, there are a number of unplugged games, interactive books, and manipulative sets that can help you transform your classroom learning experience. Some of these include the following:

- Let’s Go Code Activity Set by Learning Resources (LER2835)

- On the Brink game by ThinkFun (52123651)
- My First Coding Book: Packed with flaps and lots more to help you code without a computer! by Kiki Prottsman
- Rover Control game by ThinkFun (EZ112822)
- CodeMaster Programming Logic Game by ThinkFun
- Future Coders Cube Stackers by Alex (890110)

Completing the Requirement

You may complete this requirement either by linking to an existing coding activity, describing an existing activity you find within instructions for a game or coding manipulative set, or even creating your own unplugged activity. Whichever option you choose, you must justify two criteria:

1. The activity teaches a coding concept from this badge (i.e., commands, loops, nested loops, events, conditionals).
2. The activity is developmentally-appropriate within your teaching domain.

The first criterion is straightforward. Simply state the coding concepts the activity teaches.

The second criterion is a little more subjective and will therefore require sound argument and evidence in your written justification. To help you determine whether an activity is developmentally-appropriate within your teaching domain, use what you already know about child development and pedagogical best practices to provide a few reasons why you believe the activity would be a good one. Among others, you might consider the following factors:

- **Physical Development**: Do the students have the motor skills they need to complete the activity successfully?
- **Cognitive Development:** Does the activity present the material in a way that bridges their existing knowledge with the new knowledge?
- **Affective Filter:** Will the students enjoy doing the activity?
- **Best Practices:** Does this activity align with best practices for teaching in your target domain?

Requirement 2.2: Codable Robot Learning Activity Plan

Create an activity to guide learning with the codable robot of your choice. The activity should meet the following criteria:

Learning Activity Criteria:

- *Is developmentally-appropriate for students within your teaching domain.*
- *Helps students learn or extend their learning of one or more of the coding concepts listed in requirement 1.1 (i.e., while loops, for loops, variables, functions, parameters, operators).*
- *Helps students make connections between the coding and/or computational thinking principles they are learning and other academic content (e.g., science, math, engineering, language arts, social studies, etc.).*

On your submission form, include the following:

Submission Elements:

- Robot Used: *An indication of which robot this lesson plan is for. This should be the same robot for which you evaluated an activity in requirement 2.2.*
- Computational Learning Challenge: *A challenge that you would issue to students in order to guide their learning.*
- Scaffolding Artifacts: *A description of artifacts (if any) that you would give to students to scaffold their learning (e.g., worksheets, physical objects).*
- Student-Teacher Interaction Plan: *A description of how you would interact with the students in order to foster learning (e.g., what are you doing while they are working? What support do you provide when they get stuck?).*
- Solution Key: *The specific code (or an example of specific code) that would satisfy the computational learning challenge.*
- Criteria Justification: *A written justification for how the activity meets the criteria listed under the "Learning Activity Criteria" section above.*

Information and Resources for your Activity Plan

Robot Used

You may use any codable robot for this activity. At the time of publication of this badge guide, the following robots are available for checkout from the BYU McKay School Tech Lab for all BYU students and teachers at BYU's partner schools. Additional robots may be available when you are completing this badge. You can see the current list of

available robots and learn how to reserve them by following this link (http://bit.ly/2HWCpv7). You can also visit the Robot Guides page (http://bit.ly/2L2bXBf) on the Tech with Kids website (http://bit.ly/2IO8wNL) to learn more about each of the following robots.

- Bee-Bot
- Ozobot
- Dash
- Sphero Sprk+

Computational Learning Challenge

The computational learning challenge is a coding task that the students need to do to get the robot to perform a certain action. Sometimes it may be accompanied by specific requirements, such as a limit to the number of blocks of code that students can use, or a requirement that they use a loop. Here are some examples:

- Make Sphero draw a square with no more than 4 lines of code.
- Get Dash to do a dance. Make sure you use a loop.

Scaffolding Artifacts Description

A scaffolding artifact is anything that you will use to help the students learn the content they need to complete the learning challenge. For example, will they use a worksheet? Will there be physical objects like an inclined plane or obstacles that they need to navigate through? For this badge, you do not need to create the scaffolding artifacts, but you should describe what they would be and how you would use them in the activity.

Student-Teacher Interaction Plan

Describe how you will interact with the students. How, for example, will you give instructions for them to complete the learning activity? How will you provide the necessary scaffolding? How soon will you intervene when you see students struggling? How will you intervene? As you create your student-teacher interaction plan, keep in mind the following research-based best practices for teaching coding through robotics (Hunsaker, n.d.):

- **Modeling:** Teachers should set an example of learning by modeling their own understanding, learning, and progress in computational thinking. Especially in the early stages, they should also model the computational thinking process for students so they understand what the learning, reflection, and revision look like (Highfield, 2015).
- **Integrating:** Teachers should collaborate with other teachers to facilitate the completion of interdisciplinary culminating projects (Bers, Flannery, Kazakoff, and Sullivan, 2014).
- **Releasing Responsibility Gradually:** When teaching CT, educators should start with direct instruction, move to a simple guided activity, then issue an open-ended challenge or problem (Buss and Gamboa, 2017). Teachers should then continue to guide behavior, even while working/playing as a team (Highfield, 2015).
- **Encouraging:** Insofar as possible, teachers should provide "encouragement and problem-solving hints and tips," rather than outright answers (Buss and Gamboa, 2017).
- **Questioning:** Rather than providing answers directly, teachers should ask "probing questions" before, during, and after learning activities (Buss and Gamboa, 2017; See also Highfield, 2015) These questions should encourage students to reflect on their learning and might begin with phrases like the following (Buss and Gamboa, 2017):
    - "What if you were to…"
    - "How would you…"
    - "Have you considered…"
- **Fostering alternative problem-solving:** Teachers should promote alternative ways of modeling a problem (Buss and Gamboa, 2017), such as
    - Drawing out solutions on paper.
    - Discussing alternative solutions as teams.
    - Relating challenges to more familiar circumstances.
- **Using CT vocabulary across the curriculum** (Yadav, Mayfield, Zhou, Hambrusch, and Korb, 2014): This can reinforce students' understanding of the terms and

help them see their applicability across the curriculum and in daily life. For example, a teacher might refer to a set of rules or procedures as an “algorithm”; invite students to create an “abstraction” of how they feel; or emphasize that you are practicing “pattern recognition” skills.

Solution Key

The solution key is the specific code (or an example of specific code) that would satisfy the computational learning challenge. The code may be in whatever language makes the most sense (including a block-based language, or even just plain English that describes the blocks and their relationship). You might consider building the code on the mobile device that controls the robot, then taking a screenshot of it to fulfill this requirement.

Criteria Justification

Please note that this requirement refers to the criteria listed under the bold “Criteria” heading in the badge requirement. You should already be familiar with these criteria from completing the Teaching Early Coding Level 1 badge (http://bit.ly/2wlQjFN). If you need help with this requirement, please refer to the documentation for that badge.

Requirement 2.3: Pedagogical Reflection

On your submission form, write a reflection in which you thoroughly respond to each of the following prompts:

1. *What features of the codable robot you chose most appeal to you? Why?*
2. *In what ways could using this robot to teach coding in your classroom replace, amplify, and/or transform traditional classroom practices and settings? How could you use this tool to help students move from passive to interactive and creative uses of the technology? (See PIC-RAT framework.)*
3. *Name at least three (3) specific computational thinking (CT) skills, attitudes, or approaches that you anticipate your codable robot learning activity plan (requirement 2.2) would foster. For each, justify thoroughly why you think that aspect of CT is supported by your plan.*

Prompt 1: Robot Features

To answer this prompt, consider paying special attention to why the robot of your choice is particularly fitted for use in your teaching domain.

Prompt 2: PIC-RAT

The PIC-RAT evaluation model is a heuristic that facilitates the evaluation of any educational technology intervention. If you are not familiar with the PIC-RAT model, or if you need a refresher, please refer to the image below. (For additional information, please see Chapter 1: Effective Technology Integration in Royce Kimmons' open textbook K-12 Technology Integration [https://k12techintegration.pressbooks.com]).

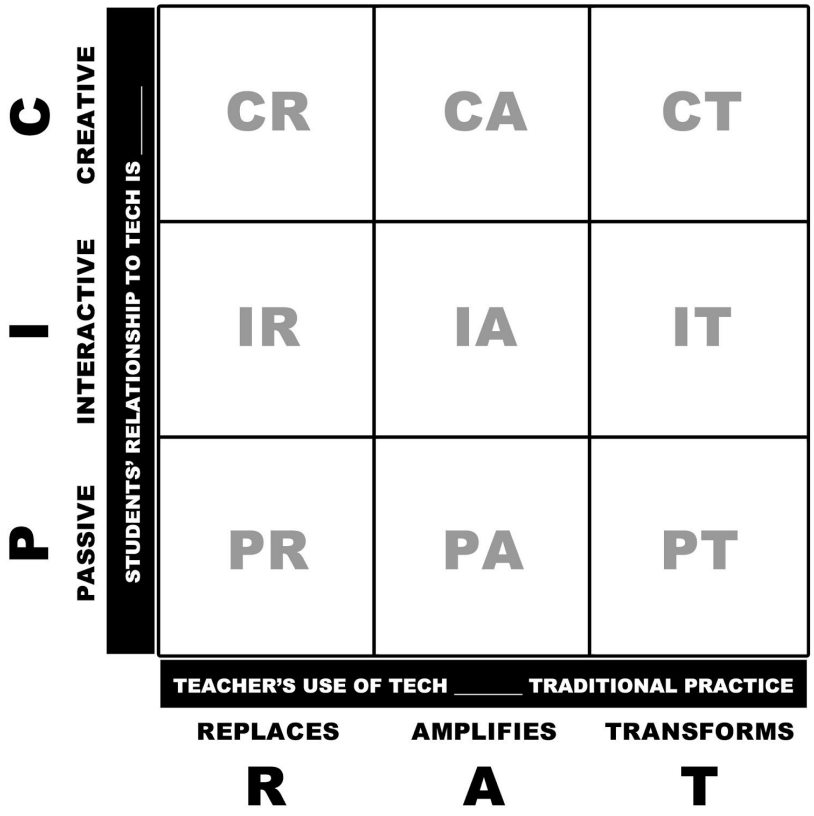


Image 3. PIC-RAT Matrix. Image by Royce Kimmons, CC-BY license.

Prompt 3: Computational Thinking Connections

If you're unfamiliar with or need a refresher on computational thinking, refer to the information provided earlier on Table 2 "Components of Computational Thinking."

References

Bers, M.U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering : Exploration of an early childhood robotics curriculum. *Computers & Education, 72*, 145–157. https://doi.org/10.1016/j.compedu.2013.10.020

Buss, A., & Gamboa, R. (2017). Teacher transformations in developing computational thinking: Gaming and robotics use in after-school settings. In P.J. Rich & C.B. Hodges (Eds.), *Emerging research, practice, and policy on computational thinking* (pp. 189-203). Cham, Switzerland: Springer. Retrieved from http://sci-hub.cc/downloads/1d8d/10.1007@978-3-319-52691-1.pdf

[Code.org] (2013, February 6). *What most schools don't teach* [Video File]. Retrieved from https://youtu.be/nKIu9yen5nc

Highfield, K. (2015). Stepping into STEM with young children: Simple robotics and programming as catalysts for early learning. In C. Donohue (Ed.), *Technology and digital media in the early years: Tools for teaching and learning* (pp. 150–161). New York, NY: Routledge.

Hunsaker, E. (n.d.). Integrating computational thinking. In R. Kimmons (Ed.) *K-12 technology integration.* Pressbooks. Retrieved from https://k12techintegration.pressbooks.com/chapter/integrating-computational-thinking/

Kimmons, R. (n.d.) Effective technology integration. In R. Kimmons (Ed.) *K-12 technology integration.* Pressbooks. Retrieved from https://k12techintegration.pressbooks.com/chapter/effective-technology-integration/

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education (TOCE)*, 14(1), 5.