Professional Development Situation: Training
Skill Focus: Enabling Active STEM Learning
Time Required: 120 minutes

BREAK IT DOWN

Participants will learn how breaking problems into smaller, manageable parts is key to computational thinking and build this skill.

Agenda

Welcome and introductions – 5 minutes
See the skill in action – 5 minutes
Practicing computational thinking – 25 minutes
Process the experience with a gallery walk – 10 minutes
See the skill in action – 10 minutes
Using a graphic organizer – 10 minutes
Engaging in active learning at the computers – 40 minutes
Process the experience – 10 minutes
Conclusion – 5 minutes

Materials

- Computer with internet connection (1 per two participants) and speakers
- Paper and pens/pencils (note taking)
- Name tents – preprinted (recommended) or blank
- Monster activity supplies (1 set per group)
    - Monster catalog
    - 3 blank monster faces
    - Markers and Scissors
- Flip chart paper, masking tape to hang paper, and markers
- Step-by-step activity
- Handouts

- Defining Computational Thinking (1 per participant)
- Graphic Organizer (1 per participant)
- Monster Catalog (1 per participant) – pages 7-23 of Lesson 3: Computational Thinking (click on open lesson plan to download)

Before the Session

- **Read this training guide** to familiarize yourself with the content and personalize the activities to best suit your style. Watch all videos and read informational materials.
  - *Italics indicate text that can be read aloud or emailed to participants.*
- Send reminder email about the meeting. Determine if any participants require accommodations (sight; hearing; etc.). If needed remind them to bring their own.
  - *The next professional development opportunity to enhance our STEM skills will be on DATE at TIME at LOCATION. Our topic for this session will be "Break It Down." Let me know if you require any accommodations. I am happy to answer any questions you have and look* forward to seeing you at the workshop. I can be reached at CONTACT INFORMATION.
- Review the Code.org activity Lesson 3: Computational Thinking (click on open lesson plan to download).
- Gather all materials and make copies of the handouts.
- Preprint name tents (recommended) or have blank name tents ready.
- Develop a list of questions participants might have. Create potential responses to be explored through informal conversation. Review any items that may be unclear.
- Test the audio and video equipment.

Training Outline

Welcome and Introductions (5 min)

- Greet participants as they arrive. Make sure everyone feels welcome and comfortable. Point out restrooms, refreshments, etc.
- Have participants pick up preprinted or blank name tents and, if needed, write their name and program on them.
- Introduce yourself and the topic: "Break It Down".
- If you have 12 or more participants, divide into groups of four and have each person introduce themselves to their group. If you have less than 12, have each person introduce themselves to the whole group in 30 seconds or less.

See the Skill in Action (5 min)

- *Today we will focus on strategies that will actively engage young people in computer science. We're going to start with an unplugged activity from Code.org to practice. Unplugged activities are fun computer science experiences that reinforce important skills but do not require a computer. I'm going to introduce this unplugged activity with a video.*

- Watch the video, "Computational Thinking". (1:34 minutes)
  - *There was some vocabulary in the video that may be new to you. She mentioned four steps of computational thinking. We will be using these terms today, so I want to be sure we are all thinking of them in the same way. (Distribute the "Defining Computational Thinking" handout.) If your group is not familiar with computational thinking, use these steps to introduce the vocabulary with participants. If they are familiar with the term, ask someone to summarize what computational thinking is and why it is important for young people.*
  - *ISTE and CSTA define computational thinking as "a problem-solving process that includes formulating problems so that we can use a computer or other tools to help solve them. It includes logically organizing and analyzing data; representing data through abstractions such as models and simulations; automating solutions through algorithmic thinking and generalizing the solution to a wide variety of problems."*
  - *The four steps or practices of computational thinking were explained in the video and they are also on your handout. Do you want us to revisit any of these terms or how they are related to this activity?* (Pause for input, participants must understand that the vocabulary words refer to the practices or steps that learners use in computational thinking.)
- Your participants will **decompose** the problem as they identify the different parts of the task and create a game plan.
- Then they will look for **patterns** that are shared by the monsters in the catalog.
- **Abstractions** will help them focus on what parts of the pattern are true for all the monsters: like they all have eyes and what differs from one monster to the next like the type of eyes.
- Then they should be ready to write a simple set of instructions, or an **algorithm,** that another group can follow to create the monster they've described.

Practicing Computational Thinking (25 min)

- Prepare for the Code.org activity Lesson 3: Computational Thinking (click on open lesson plan to download). Divide participants into groups of two. Be sure each group has: one monster catalog, three blank monster faces, markers, scissors, glue stick and a piece of flip chart paper.
    - *We have been asked to help identify monsters found on the planet Zuron. Our computers need to be able to generate images of the monsters based on eye-witness accounts. There are quite a few monsters to describe, and it may seem challenging, but luckily, we have some tools that will help us.*
    - *We are going to use computational thinking to make this task easier. The four practices of computational thinking can be used to make difficult problems easier to solve.*
    - *First, we will decompose the problem—we are not just talking about zombies here. Instead, we're talking about breaking a big, bad problem down into something much simpler. Often, big problems are just lots of little problems stuck together.*
    - *Then we will look for Patterns—sometimes, when a problem has lots of little pieces, you will notice that the pieces have something in common. If they don't, then they may at least have some striking similarities to some pieces of another problem that has been solved before. If you can spot these patterns, understanding your pieces gets much easier.*
    - *Once you recognize a pattern, you can "abstract out" (ignore) details that make things different and use the general framework to find a solution that works for more than one problem.*
    - *When your solution is complete you can write it up as an algorithm, or instructions that allows it to be processed step by step, so that the results are easy to achieve.*
    - *Now I'm going to give your groups 20 minutes to work through this activity on your own to create your algorithm and test it with another group. As you work through the process, create a list of the tasks you do and write them on the flip chart paper.*
- Let participants struggle if necessary. After about 15 minutes, or as groups finish, remind them to test their algorithms with another group.
- Give the group a five-minute break as you gather the flip charts and hang them up together for a gallery walk.

Process the Experience with a Gallery Walk (10 min)

- *Now I want you to take five minutes to look at our different charts. What are the patterns? Focus on what is similar and what is different. Then we will discuss them. (Allow 5 minutes)*
- *Let's talk about what you noticed about the charts:*
  - *What were the tasks that appeared on all the charts?*
  - *What were the tasks that were only on one or two charts?*
  - *What tasks made this an active learning process?*
  - *Did some charts have the tasks listed in a different order?*
  - *Were these differences important? Why or why not?*
- *The activity you just did focused on components within each of these parts of computational thinking, which computer scientists call aspects. However, it is important to note that what we just learned are not the only components of these different aspects. For example, abstraction also includes removing non-essential or non-relevant information in relation to the task you are trying to do. The components you have learned today are the beginning to understanding the broader aspects of computer science.*

See the Skill in Action (10 min)

- *I am going to share a video of a group of youth doing the monster activity we just did.*

- Watch the activity overview video, Computational thinking with monsters
  - *What did you notice in the video? Share your observations in the chat box.* (Pause so participants can type.)
  - *Now let's watch a second video that focuses on active learning. As you watch the video, notice what the facilitators do to make the activity successful. Pay attention to how they talk about computational thinking.*
- Watch the skill video, Breaking Down Active Learning
- Use these questions to facilitate a discussion in the chat box about the videos.
  - *How does Dagen engage youth in active STEM learning?*
  - *How did the questions Dagen ask youth help facilitate active STEM learning?*
  - *How do you know if youth are engaged in active STEM learning?*
- You can also discuss the questions as a group.

Using a Graphic Organizer (10 min)

- *For this next activity, we're going to work in groups of three or four. You will use this graphic organizer to reflect on two things about teaching computational thinking. 1) How can you keep it an active learning experience? 2) How can youth apply these thinking practices to their real world? I'll give you until* ____ (time in 10 minutes) *to work in your group.* (Encourage participants to move and divide into groups as you distribute the graphic organizer handout.)

- If time allows, go through each section of the graphic organizer and invite groups to share one idea for each practice.

Engaging in Active Learning at the Computers (40 min)

- *Now on our computers we are going to try out a curriculum from Harvard Graduate School of Education called Creative Computing which uses Scratch to teach a variety of concepts.*
- *Scratch is a visual programming language designed by MIT. How many of you have heard of Scratch? Has anyone used it?* Pay attention to who raises their hand and try to place someone who has used Scratch before with someone who hasn't. *If no one has used it you may need to explain how Scratch works a little more at the beginning of this activity.*

- Get everyone situated with a partner at a computer.
  - *The teaching model we are using works well with a variety of learners. Once we get to the activity, we'll start out working through it step by step together and move toward you working on your own with your partner. I'll be here to answer questions, but I'm not an expert. If you and your partner are ready to go on your own, feel free to do that.* (Distribute Handout "Debug it!")
  - *The Creative Computing curriculum has six units that you can go through sequentially, but today we're going to jump into unit one to an activity called Debug It! Who can explain what debugging means?* (Pause for response from the group.)
  - *Debugging is figuring out what's wrong in a computer program—it is an important skill to learn in computer science. These practices we've been using today are helpful in debugging, so think about decomposition, patterns, or abstractions that can help you figure out what the problem is.*
- Have participants go to http://scratch.mit.edu/studios/475483. This activity is on page 35 of the Creative Computing curriculum.

- *There are five projects in this studio. We'll do the first one together, and then you can work through as many of the others as you have time for. Click on Debug It 1.1. There are instructions on the right side of the screen.*
- *Read the instructions, then click on the green flag to see what Scratch and Gobo do. Click on the red stop sign to end the program.*
- *Click on the "See Inside" button at the top right-hand side of the screen and look at the code that makes each sprite move. First look at Scratch's code because you know that it's running correctly.*
- *What do you see?* (e.g., different colors and words, different shaped boxes that are different colors, or different blocks that fit together).

- If you have participants that have never used Scratch before, say:
  - *On the left side of the screen there is a tab at the top that says "Code", and in that tab are several different colors on the left side of the screen. These tell you how that code will affect your sprite. For example, blue stands for motion, so when you use motion code, it will cause your sprite to move. If you want your sprite to make a sound, you would use magenta code. There are Scratch tutorials located in the blue navigation bar at the top.*
  - *When you understand the basics of this code, click on the Gobo sprite button at the bottom of the screen to look at the code for this sprite. Try to figure out what is wrong with the code that is causing Gobo to work incorrectly.*
  - *When you think you know what is wrong with the code, make the change and see if it works. Then move on to the next project.*
- Move around the room to help people, but try not to give them answers. Instead ask questions to help them think about solutions. Use the terms decomposition, patterns, abstraction and algorithm as appropriate.

Process the Experience (10 min)

- Bring the group back together. If possible, move away from the computers for discussion.
  - *As we gather to process our experience debugging, I want to point out a couple of things that you will find helpful as you are facilitating computer science activities. First, moving away from the computers makes it much easier for me to get everyone's attention for this discussion. Second, as we did this activity, I tried to model active learning strategies that you can use, so let's think about how that worked as we process the experience. I'm going to start thinking about how the*

experience made you feel because emotions are very important in preparing students to learn.

- Use these questions to guide the discussion and process this experience.
    - *How did you feel about debugging?*
    - *What made you feel frustrated? How did you manage that?*
    - *How did you feel when you figured out what was wrong with the code?*
    - *How would learning how to debug a program help the young people you work with?*
    - *How did you use your understanding of decomposition, patterns, abstraction or algorithms as you were debugging the programs?*
    - *How did understanding computational thinking help you debug?*

Conclusion (5 min)

- *Thank you for coming to the session today. We learned about decomposition, breaking problems into smaller, more manageable parts and saw how this practice is key to computational thinking. We also focused on strategies that you can use to actively engage young people in computer science. Remember to integrate the ideas you put on your graphic organizer when you get back to your program. Following the session, I will share links to the monsters activity and the Creative Computing curriculum so you can use them to practice the skills you developed today.*

- Answer any final questions participants may have.

After the Session

- Within three weeks of the training, send an email to participants that includes links to the activities used in the workshop:
    - *Thank you for your participation in the recent Click2Science training on "Break it Down". I hope you found it useful. Consider meeting with a co-worker, supervisor, or friend to share what you learned. I look forward to continuing our learning at the next session on SKILL/FOCUS on DATE at TIME at LOCATION. Please let me know if you have any questions. I can be reached at CONTACT INFORMATION.*

      Computational thinking activity link: https://studio.code.org/s/20-hour/stage/3/puzzle/1
      Creative Computing link: http://scratched.gse.harvard.edu/guide/index.html

Want to Earn Credit? Click2Science has teamed up with Better Kid Care to provide continuing education units. Check it out at: http://www.click2sciencepd.org/web-lessons/about

This material is based upon work supported by the National Science Foundation under Grant No. 1840947

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Defining Computational Thinking

The International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA) have collaborated with leaders from higher education, industry, and K–12 education to develop an operational definition of computational thinking. The operational definition provides a framework and vocabulary for computational thinking that will resonate with all K–12 educators

Computational thinking (CT) is a problem-solving process that includes (but is not limited to) the following characteristics:

- Formulating problems in a way that enables us to use a computer and other tools to help solve them.
- Logically organizing and analyzing data
- Representing data through abstractions such as models and simulations
- Automating solutions through algorithmic thinking (a series of ordered steps)
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources
- Generalizing and transferring this problem-solving process to a wide variety of problems

The definition for computational science listed above can be found at: http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf

Decompose describes the thinking practice of breaking a problem down into smaller pieces.
Pattern Matching describes the thinking practice of finding similarities, or patterns, between things.
Abstraction describes the thinking practice of pulling out specific differences to make one solution work for multiple problems.
Algorithm is a list of steps that you can follow to finish a task or reach a solution.

Graphic Organizer

Decomposition Breaking down a large problem into smaller pieces	**Pattern** Finding a theme that repeats several times
Abstraction Removing details from a solution so that it will work for a lot of different problems	**Algorithm** A list of steps used to complete a task