Using Computer Science to Think Through Word Problems

K. Ann Renninger, Quinn A. Basewitz, Sammi K. Chai, and Alexandra S. Fischmann
Department of Educational Studies, Swarthmore College, Swarthmore, PA

Corresponding author:
K. Ann Renninger
Dorwin P. Cartwright Professor of Social Theory and Social Action
Professor, Department of Educational Studies
Swarthmore College
krennin1@swarthmore.edu
p: 610-328-8347
https://orcid.org/0000-0002-3054-4637

Funding:

This research was funded by the National Science Foundation grant #1738606; any opinions, findings, and conclusions or recommendations in this article do not necessarily reflect those of the National Science Foundation.

Acknowledgements:

We gratefully acknowledge comments on earlier drafts of this research from Emmanuel Schanzer; and Jane Huynh's editorial support.

Using Computer Science to Think Through Word Problems

Many students find word problems difficult, and it can be hard to figure out how to convince them to consider what a problem is asking before jumping into calculations. Working with computer science tools can encourage students to focus on engaging with the components of word problems, as well as their context. The integration of mathematics and computer science provides students with an approach to problem solving that is not simply procedural.

Our work studying teacher professional development is showing that integrating computer science into math lessons supports students (advanced, as well as those with learning differences, or who are ELL) to "see" how they can break problems into parts. We are finding that integrating math and computer science can support students to think about the math in the problems they are assigned (e.g., recalling earlier math concepts when approaching a new problem), and typically does not require re-teaching.

Integrating mathematics with computer science tools is relatively simple and exciting, as it can support the development of students' computational thinking (breaking down problems, recognizing patterns, designing algorithms, and using logical reasoning to solve them). "Integration" does not involve simply using computers in the math classroom, but rather leveraging computer science tools to strengthen mathematical understanding, especially of functions. Integrating mathematics and computer science provides students with support to engage with mathematics as sense-making– it positions them to not only think through word problems but to think about math conceptually.

A number of integration tools exist (e.g., Computer Science Integrated with Mathematics in Middle Schools' [CSIMMS] modules; Computing, Science, Technology, Engineering, and Mathematics' [C-Stem] robotics program); here we focus on sharing information about Bootstrap Algebra's tools (https://www.bootstrapworld.org/materials/algebra/), because these target the development of students' conceptual understanding of word problems, and abilities to work with them. In the tables that follow, we

provide an overview of four types of tools, examples of their use by teachers and students, and relevant links. Table 1 describes Contracts; Table 2 presents Design Recipes, Table 3 explains the Flags Activity, and Table 4 introduces Circles of Evaluation.

Table 1:

Contracts and Lesson Card

Computer science tools can make word problems more approachable for students, and are not difficult for teachers to use. For example, teachers' use of the Contracts tool encourages students to think about a problem's structure (bit.ly/BootstrapContracts) — computer science enables students to see the big picture, its individual components, as well as their relations. "Contracts" can be used multiple times with different problems, and supports students to both explore and identify the components of functions: the name, the domain/input, and the range/output. A glossary of computer science terms is provided in Figure 1, with examples from work with Contracts.

Contracts Lesson Card

Objective of Activity: ● Students will be supported to work with functions by having the vocabulary of name, domain, and range. **Materials:** ● Chromebooks
Teaching Contracts:

The teacher can start their lesson using the Bootstrap example which compares functions to machines that do three things, in the following order:

- It takes in some value(s) (input/argument)
- It does something to that value(s) (e.g., multiplies, counts, creates an image)
- It produces a new value (the output) (bit.ly/BootstrapContractsLessonSlide3)

Students are instructed to type specific lines of code into the code editor and to observe the output as they change the input to each function. For example, Stephanie Harrison, a teacher at online Connections Academy, used the following sequence with her sixth graders (note: a brief glossary of computer science terminology used here is provided in Figure 1):

- Type num-sqrt(16) into the Interactions Area.
- What part of this expression is the value?
- What is the name of this function?
- How many arguments are we giving to this function?
- What is the type of argument we are giving to *num-sqrt*?
- What did *num-sqrt* produce?

Students worked together to address each of these questions and showed each other how they solved and/or dealt with errors. Stephanie started with the sample function above and then had her students work with additional functions.

In this lesson, students learned to recognize Contracts, the name, input, and output of computer science functions in the following format: *name: domain -> range*. This tool can be reused in other math contexts as students identify important aspects of various functions.

Shay Crawford, a teacher at urban Longfellow Middle School, reported that the Contracts lessons set most students up to independently explore programming (and math) with small substitutions

in their code (e.g., making the target blue, instead of green). This leads them to additional exploration and questions, and there is always enough material to fill up class time.
Potential Extensions
Connecting Contracts to Function Composition ● Students can be supported to explore and discuss more complex functions, where two functions can be combined by using the output from the first function as the input to the second function.
Connecting Contracts to Inverse Functions ● Students can explore inverse functions and consider how the domain and range relate to an original function.

Figure 1:

Computer Science Glossary

Pyret	A coding language with an online editor used with Bootstrap curriculum.
function_name(input)	Functions in computer science are written like functions in math, f(x). The function name is on the outside of the parentheses, and the arguments, or inputs, are inside the parentheses.
arguments	Inputs to the function (these are separated by commas when there are multiple arguments). These are similar to parameters.
num-sqrt(16)	This function call outputs the square root of sixteen.

string-length("rainbow")	This function call outputs the number of characters in the word rainbow.
triangle(20, "solid", "red")	This function call outputs a triangle image of size 20 that is solid and red.

Table 2:

Design Recipe, Lesson Card, Matching Activity

The Design Recipe (bit.ly/BootstrapDesignRecipe) is a tool that builds on Contracts. Design Recipe can be used repeatedly with different and more complicated word problems to support students' developing abilities. It guides work with abstraction; supporting a deepening understanding of functions. Design Recipe includes a Contract (the name, domain, and range), a purpose statement that describes what the function does, examples to verify that the function definition works, and a function definition written in general form. As Shay Crawford observed, Design Recipe helped his students to "do sense-making, and pay attention to word problem components and their context." Shay really liked that Design Recipe offset students' tendencies "to rush and/or engage in ineffective problem solving."

Lesson Card

Objective of Activity:

- Students will be supported to break down word problems into their components using the Design Recipe tool.
- Students will recognize the purpose of the functions they are working with.
- Students will write purpose statements in their own words.

Materials:

- Chromebooks

- Rocket Height Pyret Starter File (bit.ly/BootstrapDesignRecipesRocketStarterFile)

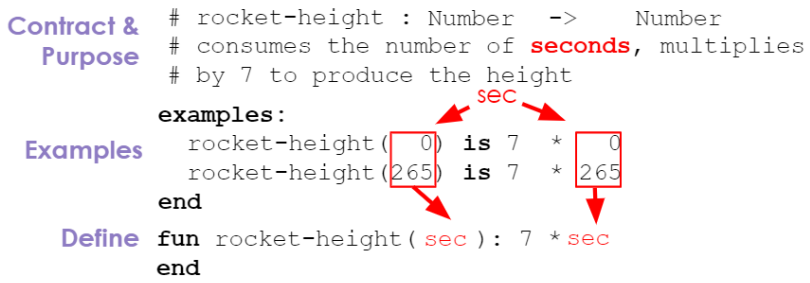
Teaching Design Recipes:

Design Recipe builds on Contracts by including a purpose statement, examples, and a function definition. The purpose statement uses everyday language to explain:

- what the function takes in,
- what it produces, and
- what is involved in getting from the input to the output.

In Bootstrap's introduction to the Design Recipe tool, a scaffolded version of the Rocket Height program leads students to write a function that controls the height of a rocket that is moving at 7 units per second. A built-in interactive tool leads the students to see how their work plays out visually.

In this activity, the students create the following Design Recipe:



The Contract and purpose statement are written as comments. This means that the computer does not read them as code, since they have a # symbol at the beginning of each line. Through the use of comments, the student can include their everyday language as well as their code all on the same screen. Contracts follow the format explained in Table 1: *name: domain -> range*. The purpose statement describes what the function does. In this example, students are calculating rocket height where the rocket moves at a constant speed, 7 units per second. The function takes in the number of seconds that have passed, multiplies that number by 7, and outputs the height of the rocket.

The examples section includes mini-tests that lead students to check that the function definition works. Examples support students to have a visual reference and be able to see which terms are constants and which are variables.

The function definition is the actual code that Pyret uses when the function *rocket-height* command is run. The function is written in a generalized form; this takes in and uses the number of seconds, and is referred to as the variable *sec*.

From here, there are multiple activity options.

1. Matching Word Problems and Purpose Statements (bit.ly/BootstrapDesignRecipesMatching)
   An example from Shay Crawford's instruction is included in Table 2a.
2. Writing Examples from Purpose Statements (bit.ly/BootstrapDesignRecipesExamples)
3. Fixing Purpose Statements (written by ChatGPT) (bit.ly/BootstrapDesignRecipesFixing)

Potential Extensions

Connecting Design Recipes to geometry

- Students can use Design Recipe to explore functions in geometry that calculate geometric properties such as area, perimeter, or volume.

Matching Word Problems and Purpose Statements

This matching activity includes a set of word problems each of which includes the name *Rex*, and a set of prewritten purpose statements.

An example of one of the word problems reads: *"Annie got a new dog, Xavier, that eats about 5 times as much as her little dog, Rex, who is 10 years old. She hasn't gotten used to buying enough*

dog food for the household yet. Write a function that generates an estimate for how many pounds of food Xavier will eat, given the amount of food that Rex usually consumes in the same amount of time."

Shay made this a whole class activity, and began by prompting the students: "If Rex eats one pound of food, how much does Xavier eat?" As students offered answers, he wrote them vertically on the board:

1 * 5 = 5

2 * 5 = 10

3 * 5 = 15

By stacking the examples in this way, Shay supported the students to recognize that the input variable was the amount that Rex ate and that it was multiplied by 5 to get to the output: the amount that Xavier ate. Students then matched this problem to its purpose statement, drawing from examples such as: *"Consume the pounds of food Rex eats and add 5," "Consume the pounds of food Rex eats and multiply by 5.*" The students decided to use the variable *r* to stand for how much Rex eats and *x* to stand for how much Xavier eats and wrote the generalized function definition: *r * 5 = x.*

Table 3:

The Flags Activity, Lesson Card, Rainbow Activity, and Student Examples

The Flags Activity (bit.ly/BootstrapFlags) is yet another example of an activity that models how computer science can be used to develop students' skills in working with problems that they haven't encountered before. The design of a flag can be relatively simple, or quite complex. In this hands-on activity, students are heavily scaffolded at first to recreate the Japanese flag (because of its simplicity) using code. Given an image of the flag, the task is to identify the components of the flag's design and then to build the flag using the components they defined. The website provides a range of flags that can then be replicated, and students can also create their own flags. Working with this activity enables students to hone their skills, and attend to and work within the context of problem components.

Flags Activity Lesson Card

Objective of Activity:

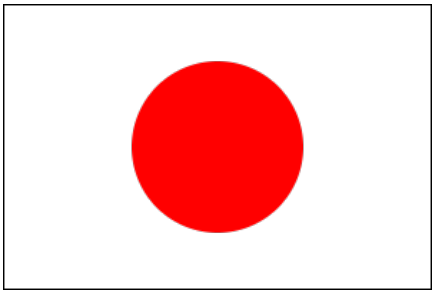
- Students will decompose a composite image into its component parts.
- Students will create composite images using functions and spatial reasoning.
- Students will dilate and translate shapes on a coordinate plane.

Materials:

- Chromebooks
- Flags Activity Starter Pyret File (bit.ly/BootstrapFlagsStarterFile)

Teaching the Flags Activity:

The Flags Activity is an introduction to functions and the coordinate plane and involves editing code to first create the flag of Japan.



The activity is broken into four parts: defining the shapes (creating the background and the dot), describing the image (creating a description of the image that students will build), building the flag (creating a new variable, Japan, that combines the two previously defined variables), making adjustments (comparing the image they created to the goal image and making adjustments).

Students begin by opening the Flag Activity starter file on Pyret, where they are provided with instructions and code that allows them to work through the activity either independently or with a small group without step-by-step guidance from the teacher.

While building the flag, students can resize the circle, and experiment with the (x, y) coordinates to place the dot in the middle of the rectangle. Each of these variable definitions uses an image function which has its own specific input requirements. This hands-on experience will be a point that teachers can continually refer back to as they introduce related topics.

The Flags Activity, like other integration tools, encourages students to break problems down into their component parts. The Japanese flag provides an initial challenge; there are additional starter files of varying difficulty. Students can also create their own flags "from scratch" once they have completed Japan.

In order to provide her sixth graders with additional practice, Stephanie Harrison created a related project, called the Rainbow Activity (bit.ly/RainbowActivity, Table 3b).

Potential Extension

Connecting Flags to students' communities and identities: creating a flag to represent their community

- Students can be supported to make personal, culturally relevant, or even interdisciplinary connections to the Flag they are making. Students can use what they learned to create their own flag. They might research how countries' flags are decided and consider what aspects of their own community should be depicted on a flag while designing their own.

The Rainbow Activity

In Stephanie Harrison's Rainbow Activity (bit.ly/RainbowActivity), students built a rainbow using a process similar to that used in the Flag Activity. Provided with four lines of code (plus lots of comments explaining parts of the code):

```
red = circle(100, "solid", "red")
```

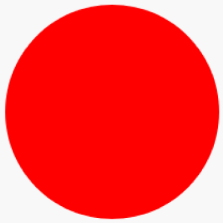
orange = circle(90, "solid", "orange")

rorain = overlay(orange, red)

*rainbow = crop(0, 0, 100 * 2, 100, rorain)*

students then defined and overlaid additional circles to continue to add colors to fill out their rainbow (see examples of students' code, constructions, and commentary in Table 3c).

Rainbow Activity Example

Input	*Output*
#CONTRACTS #circle::number, string, string -> image #The number in the circle is the radius. **#Example** red = circle(100, "solid", "red") orange = circle(90, "solid", "orange") yellow = circle(80, "solid", "yellow")	››› red  ››› orange  ››› yellow
#CONTRACTS #overlay::image, image -> image **#Example** rorain = overlay(orange, red) rorain2 = overlay(yellow, rorain)	››› rorain  ››› rorain2 
#CONTRACTS #crop:: (x :: number, y:: number, Width:: number, Height :: number, Img :: image) -> Image #Crops img to the rectangle with the upper left at the point (x,y) and with width and height. **#Example** rainbow = crop(0, 0, 100 * 2, 100, rorain2)	››› rainbow 
#Repeat functions of circles, overlay, and crop for each color of roygbiv and white to complete the rainbow image	››› rainbow 

Error Message

#An error message will pop up if the code is incorrect. Error messages provide a support for students to retrace their steps and work through mistakes.

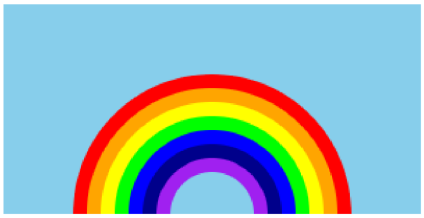
#In this example, the student passed 3 arguments (inputs) through the overlay function and received an error message that indicated that the overlay function can only accept 2 arguments.

```
1  use context essentials2021
2  ##ROYGBIV
3
4  #############
5  #Goal: to create a rainbow with a diameter
   of 100 using the colors of roygbiv using the
   functions for a circle, crop, and overlay.
6
7  ##CONTRACTS##
8  #circle::number, string, string -> image
9  ## The number in the circle is the radius.
10 ##Example
11 red = circle(100, "solid", "red")
12 orange = circle(90, "solid", "orange")
13 yellow = circle(80, "solid", "yellow")
14
15   ###overlay::image, image -> image
16 ##Example
17 rorain = overlay(orange, red)
18 rorain2 = overlay(yellow, orange, red)
```

```
This application expression errored:
                                    definitions://:17:10-17:38
18 rorain2 = overlay(yellow, orange, red)
3 arguments were passed to the operator.
The operator evaluated to a function accepting 2 parameters.
An application expression expects the number of parameters and arguments to be the same.
(Show program evaluation trace...)
>>>
```

Student Work, Rainbow Activity

Some examples of student work include:



Note: Students were able to add personalization to their rainbows. Some used their names. We have replaced those with"Rainbow" here.

Stephanie Harrison also asked her students to share their experiences about programming. For example,

- "I found that programming is an awesome way to learn from your mistakes. If you do make a mistake you can learn so much from it through the error codes."
- "Earlier in the year, when it was introduced to the class that we were going to be building a rainbow, I got a little nervous. When we started learning about strings, functions, booleans, images, it was getting easier and I was getting more confident that I could do it. When the project came up, I was ready. When I finally learned all the things about making a rainbow and made one, I was very proud of myself. I am so glad that I got to experience something like

this!”

Table 4:

Circles of Evaluation, Lesson Card, Challenge Activity

Teachers find Circles of Evaluation, or “Circles”, easy to pick up, and say that they enable their students to visualize arithmetic expressions, see purpose in calculations, and develop a shared language. Sometimes teachers feel that their students are not able to start working immediately with word problems, citing their students’ weak calculation skills and understanding of math vocabulary. These teachers think that their students’ skills can be best supported by starting with Circles (bit.ly/BootstrapCircles), as they provide a basis for thinking through order of operations. They also explain that their students start using Circles in their work, even when it is not assigned. Use of Circles is not limited to order of operations. Samatha Leonard, a teacher at urban Central Middle School, described plans to adapt Circles to use with shapes and function composition this coming year.

As part of the math sequence, Circles of Evaluation can be taught starting in elementary school. They may be used to teach a variety of topics: the commutative and associative properties, additive and multiplicative inverse properties, the identity property, the distributive property, absolute value, and exponents (bit.ly/BootstrapEarlyMath). As Kaleb Gordon, a teacher at rural Ada Junior High School, observed, having students work with Circles early on sets them up to be familiar with integration tools and ready to keep moving, “and then I don't have to worry about being like, hey, here's something completely new for you.”

Circles of Evaluation Lesson Card

Objective of Activity:

- Students will be able to see the structure of arithmetic expressions by modeling them.

- Students will be supported to see structure by using Circles of Evaluation to translate into code (see student example in Figure 2).
- Students will be positioned to work with word problems using Contracts and/or Design Recipes.

Materials:

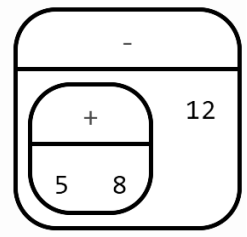
- Chromebooks
- Worksheet or Desmos Activity (included on Bootstrap website)

Teaching Circles of Evaluation:

Circles are used to help students see how they can figure out what a problem is asking. As described in the Bootstrap lesson slide for Circles, there are three rules for creating a Circle:

- Every Circle has one function, and it is written at the top.
- The inputs to the function are written in the middle of the circle.
- Circles can be written within other Circles.

When evaluating a Circle, you begin with the innermost Circle and work your way out, visually following the order of operations.



Consider the arithmetic expression 5 + 8 - 12. The Circle shows that you do addition first.

Teaching Circles might begin with a class discussion on the importance of standardization (whether that is common language, common currency, or another example), and move into a discussion of why there is a need for a consistent order of operations. Circles of Evaluation can easily be translated into code for Pyret. Because Pyret does not have built-in order of operations, it requires

students to add parentheses to tell the computer the order in which to simplify the expression. 2 + 5 * 3 would need to be coded as 2 + (5 * 3) or (2 + (5 * 3)) to follow standard order of operations. This is visually the same as the Circles of Evaluation.

There are multiple activity options for working with Circles that also offer increasing challenge. Links to two examples and details to a more CS-centered activity are included below:

1. Worksheet: Translating Arithmetic Expressions to Circles of Evaluation to Code (bit.ly/BootstrapCirclesWorksheet)
2. Matching Activity: Matching Circles of Evaluation and Code (bit.ly/BootstrapDesmosMatching)
3. Testing Code Challenge Activity: Supporting Students to Check Their Work (see Table 4b).

Testing Code Challenge Activity: Supporting Students to Check Their Work

In the Testing Code Challenge Activity, students can check to see if their diagramming of Circles is correct by translating Circles into code, debugging errors, and verifying their results using Pyret's informative error messages. Students can create tests and get results all at once when they click run.

Let's walk through an example:

A student types this check segment in the code editor and clicks run, not realizing that there is a bug (an inaccuracy):

```
check:
  (4 + 5) / 3 is 3
  ((1 + 3) / 2) is  2
  9 - 3 + 5 is 10
end
```

First, the student will get an error:

Reading this expression errored: 9 - 3 + 5 is 10. The - and + operations are at the same grouping level. Add parentheses to group the operations, and make the order of operations clear.

Pyret requires you to tell it in exactly which order to operate. The student then realizes their mistake and rewrites the check segment:

check:

(4 + 5) / 3 is 3

((1 + 3) / 2) is 2

(9 - 3) + 5 is 10

end

The code outputs the message

2 out of 3 tests passed in this block

and highlights the last test as incorrect. On paper, this student decides to draw a new Circle and reevaluates the expression to be 11, edits the code, reruns it, and passes all three tests this time, verifying their work as correct.

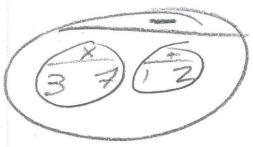
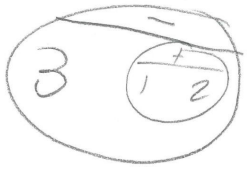
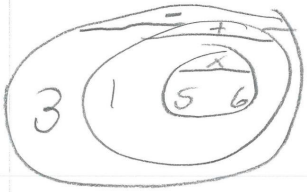
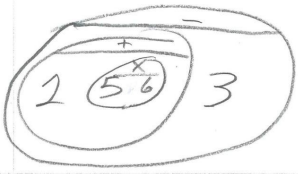
Being able to create tests to check if code is working correctly is a regular practice in computer science, and benefits math students to develop the habit of checking their work.

Figure 2

Example of Student Work with Circles

Translate Arithmetic to Circles of Evaluation & Code (Intro)

Translate each of the arithmetic expressions below into Circles of Evaluation, then translate them to Code.

	Arithmetic	Circle of Evaluation	Code
1	$(3 \times 7) - (1 + 2)$		((3×7)-(1+2))
2	$3 - (1 + 2)$		(3-(1+2))
3	$3 - (1 + (5 \times 6))$		(3-(1+(5×6))
4	$(1 + (5 \times 6)) - 3$		((1+(5×6))-3)