Investigating Concreteness Fading in the Programming Domain with an Online Computer Tutor

by

Nadia Markova

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Cognitive Science

Carleton University
Ottawa, Ontario

Abstract

Abstract concepts are difficult because students cannot connect them to prior knowledge. The pedagogical approach *concreteness fading* introduces abstract concepts through a concrete representation grounded in students' real-world knowledge. As instruction progresses, perceptual information is removed until the abstract representation is reached. This thesis investigates concreteness fading in the programming domain with university-level students (N = 50). We created a lesson to teach the concept of 'for-loops', in the Python language. The study used a between-subject design with two conditions. The experimental condition included a lesson starting with a concrete representation of the concept depicted as an animation, that gradually faded into its abstract representation, namely the Python code. The control condition included a traditional lesson with only the abstract code. The intervention was implemented with an online computer tutor built using the Cognitive Tutor Authoring Tools (CTAT) package. While learning increased overall, there was no significant effect of condition.

Acknowledgements

I have thanked everyone I could personally, but in here, I would like to thank three main groups of people again: my research supervisor, my thesis committee, and my family.

Thank you to my supervisor Kasia for absolutely everything. Thank you for being so kind and patient with my millions of questions. Thank you for helping me develop my research skills and for appreciating the way I think. I am truly grateful.

Thank you to my committee; Dr. Rebecca Merkley, Dr. Sonia Chiasson, and Dr. Deepthi Kamawar for your feedback. Thank you for making my thesis defense enjoyable (as much as realistically possible) with our discussion and your constructive ideas for moving forward with this research. I am thankful that you gave my thesis such careful attention.

And finally, thank you to my family. Thank you for your infinite support, for believing in me, and for listening to me go on and on anxiously. Thank you for your reassurance.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

In contrast to abstract representations, concrete representations may be easier for students to understand because they are grounded in the student's real-world experiences (i.e., their prior knowledge). A pedagogical approach called *concreteness fading* capitalizes on this proposal. In this paradigm, an abstract concept is first introduced through a concrete representation that links concepts to prior knowledge, and as instruction progresses, information is removed until the abstract representation is reached. Most of the prior research in this area has been conducted with mathematical concepts and young students, so work is needed to extend the findings to other domains and older populations.

To fill this gap, in this thesis we investigate the utility of concreteness fading in the programming domain with university-level students. Specifically, we created a lesson that teaches a programming concept – 'for-loops' – in the Python language. The lesson starts with a concrete representation that is gradually faded into its abstract representation, namely the Python code. The lesson was integrated into an online computer tutor built using the Cognitive Tutor Authoring Tools (CTAT) package and evaluated with a between-subject study with two conditions. The experimental condition implemented concreteness fading by introducing the concept in three stages: concrete, semiconcrete, and abstract, all integrated into the CTAT tutor; the first two stages were presented as animations developed for this thesis, demonstrating the target concept. The concrete stage provided a real-world analogy for the targe concept, while the intermediate stage leveraged this analogy with Python code. The final abstract stage corresponded to the Python code. The control condition included only the abstract stage, i.e., the Python code. Learning was measured through a pre- and posttest with standard and transfer questions. The main research

question was to investigate whether teaching this target programming concept using the concreteness fading approach would result in higher learning gains and transfer than standard instruction.

This work is one of the few studies on concreteness fading in the programming domain with university students. The created tutoring system provides a practical contribution as it is operational and can be used to teach for-loops, as well as other lessons that could also be implemented in it.

Before presenting the results, we describe the related work, the details of the lesson and the computer tutor, as well as the study.

Chapter 2: Related Work

2.1 The Concreteness Fading Approach

The goal of concreteness fading is to teach abstract concepts in a manner that is more gradual and easier to understand for students than the traditional approach which often begins with teaching the abstract concept right away. By starting with a concrete representation of the target abstract concept, and progressively decontextualizing the concrete material, the teacher can remove context-specific elements from the concrete material to lead to the abstract representation (Fyfe & Nathan, 2019). The advantage of this approach is that it helps students ground abstract thinking and unclear symbols, and understand them in terms of concrete objects that they know, can generalize from, and can refer to in future encounters with new abstract domains (Fyfe et al., 2014).

Concreteness fading was first proposed as a theory of learning by Bruner (1966), although at the time he did not give the paradigm this name. He envisioned that a concept could be taught in three stages, with three distinct modes of representation (Figure 2.1):

1. Enactive (concrete)
2. Iconic (pictorial/graphic)
3. Symbolic (abstract)

Stage 1 – the enactive stage – involves interacting with physical, 3-dimensional concrete models of the abstract concept that is being taught. The aim is to find a concrete representation of this abstract concept; a representation that is grounded in everyday life and one the student can grasp more easily than the abstract concept because it connects to their prior knowledge. For example, if the domain involves addition, the enactive stage

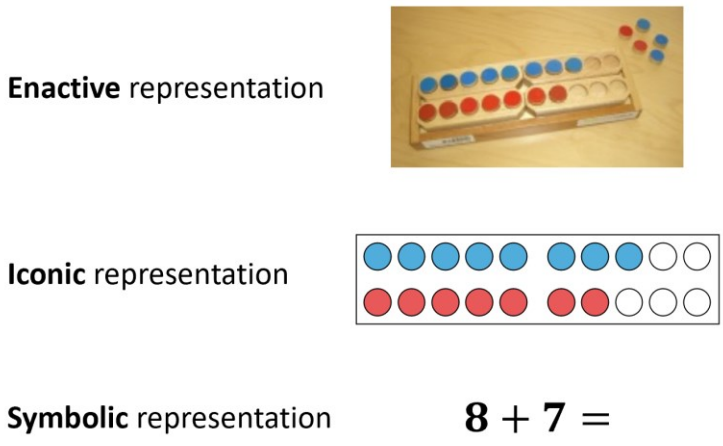
could include manipulations with pebbles, which are concrete objects that students can understand (Figure 2.1).

Stage 2 – the iconic stage – involves removing all 3-dimensional aspects of the concrete representation, leaving a graphical or pictorial model of the concept. The difference between the representation in stage 1 and stage 2 is that stage 2 is an image instead of a 3-dimensional object. For the addition example introduced above, stage 2 could now have drawings of circles representing the pebbles, instead of the physical pebbles (Figure 2.1).

Stage 3 – the symbolic stage – is the last stage consisting of the final abstract concept being taught. For the addition example, the actual numbers and symbols ('+', '=') would now be employed (Figure 2.1).

Figure 2.1

Example of Enactive, Iconic, and Symbolic Stages Used to Teach Addition



Note. Adapted from Cognitive Development. Idea to Form. (2016, October 31). Retrieved from https://idea2form.wordpress.com/2016/10/31/cognitive-development/

Bruner's model has inspired a similar instructional framework called Concrete-Representational-Abstract (CRA) widely used in mathematics education and research on students with learning disabilities (Miller & Hudson, 2007). Similar to Bruner's progression, the CRA sequence begins with a concrete physical stage, transitions to a pictorial representation, and ends with the abstract concept. Transition occurs when the student has mastered the previous stage.

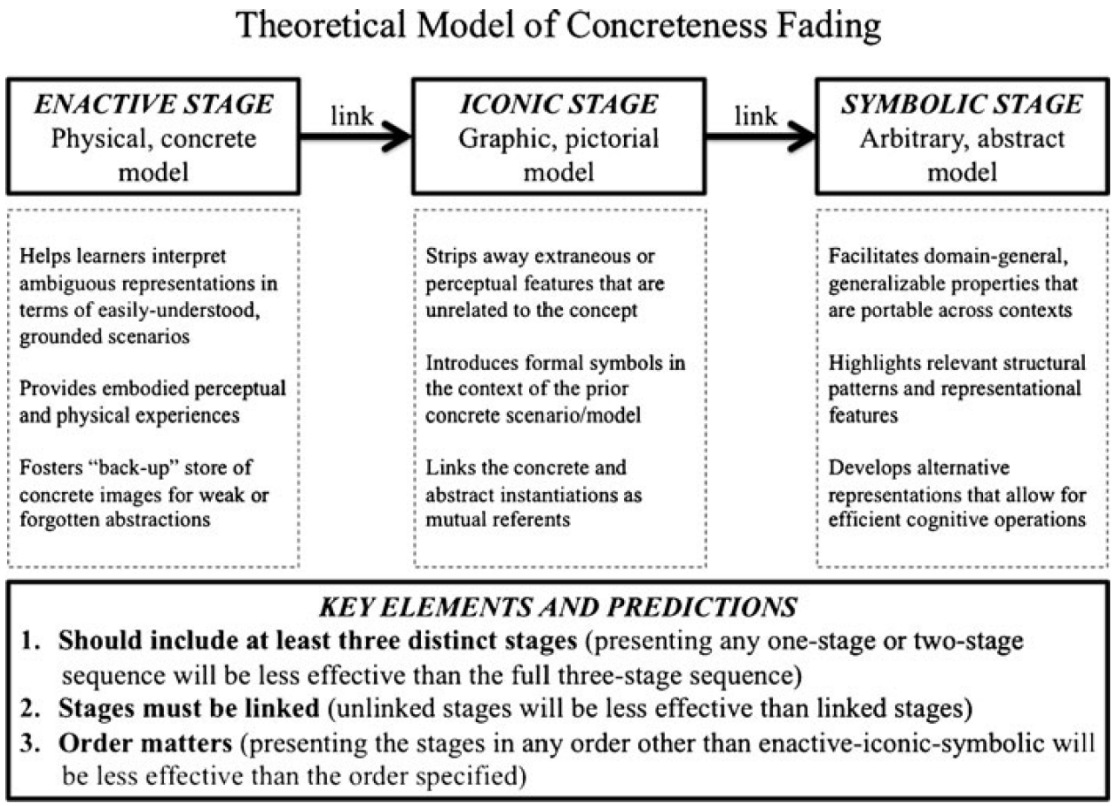
The enactive stage in Bruner's paradigm (the concrete stage in the CRA framework) involves physically interacting with the concrete representation, and "enacting" the concept with the manipulatives (for example addition). However, as will be discussed in the following sections, recent work on concreteness fading has expanded on Bruner's ideas allowing for progressions that do not necessarily have three separate stages, as well as a concrete stage that does not include interaction with physical objects.

Concreteness fading aims to help students attach meaning to abstract symbols by grounding them in concrete representations. This notion relates to Harnad's Symbol Grounding Problem (Harnad, 1990), which is a philosophical proposal concerned with how words and symbols obtain their meaning: *"How is symbol meaning to be grounded in something other than just more meaningless symbols?"* (Harnad, 1990, p.340). Thus, this proposal states that one cannot infer meaning from an abstract symbol alone without additional context. According to Harnad (1990, p. 335), *"Symbolic representations must be grounded bottom-up in nonsymbolic representations"*. Therefore, grounding symbols to some concrete representation will give abstract symbols meaning.

In concreteness fading, difficult abstract concepts are initially grounded to a concrete representation that already has meaning to the learner, which then aids them in understanding the meaning of the abstract symbols.

Figure 2.2

Guidelines for Implementing Concreteness Fading



Note. From "Concreteness Fading in Mathematics and Science Instruction: A Systematic Review" by Fyfe, E. R., McNeil, N. M., Son, J. Y., & Goldstone, R. L. (2014). *Educational Psychology Review*, *26*(1), 9–25.

2.2 Guidelines for Implementing Concreteness Fading

Researchers have proposed several guidelines for implementing concreteness fading (Fyfe et al. 2014; Fyfe & Nathan 2019). Figure 2.2 provides an overview of these principles. One guideline is that at least three distinct stages of instruction are included and presented in the order of concrete to abstract. The representations must overlap as instruction moves through each stage. It is crucial to have clear explanations as to how each stage relates to the others. The teacher (or the lesson) must be able to demonstrate the similarities between each representation so that the student can understand the links amongst them.

Concrete representations can be implemented as analogies to the abstract concept (explained in more detail in Chapter 3). For instance, an analogy to explain fractions could involve cutting a cake. If we show the notation $\frac{1}{3}$ to a child that has not learned about fractions yet, it will not mean much. More specifically, if we start explaining how the numerator on the top denotes a part and the denominator at the bottom is the whole, it may not make sense as it is abstract in the child's view and they cannot anchor it to their own experience in the real world since it is something new. But, if we introduce the fraction by making an analogy to something more concrete, for example a circle cut into several pieces, such as a cake, the child may be better able to understand the concept because they have already seen a cake in real life and they can relate to it with their prior knowledge. The analogy can also be further implemented in the form of a physical manipulative that allows the student to experience the concept grounded in the real world. Once the concept is grasped, the teacher can gradually move to the abstract notation. At this point, $\frac{1}{3}$ will now

mean something and will no longer be so abstract because the student can draw the connection to their prior knowledge when they encountered the manipulative.

In addition, although the concrete stage should be grounded in real-world experience that students can relate to, it should also avoid having too many irrelevant perceptual features. Such features can be distracting and do not add additional value to the basic understanding of the concept (Laski et al., 2015). For instance, if fractions are taught by first grounding the concepts in the experience of cutting a cake into pieces, the decorations on the cake would be irrelevant features. It is important to note that concrete does not equal physical. Something can be a concrete representation, without being a physical object. For instance, a three-dimensional drawing can be used as a concrete representation (even though it is not a physical object).

The original studies on concreteness fading (Fyfe et al., 2015, discussed in Section 2.3) as well as subsequent ones (Trory et al. 2018; Kim 2020) have been conducted using actual physical objects in the concrete stage (e.g., puppets). However, other researchers have reinterpreted the notion of 'concrete'. These latter studies still fall under the category of concreteness fading because they include three stages, but their first concrete stage is virtual rather than physical (Arawjo et al., 2017; Suh et al., 2021). These studies are discussed in the following sections.

Another guideline for implementing the concreteness fading approach is to explicitly explain the link between the concrete phase and the abstract concept. A concrete stage is useful if there is a way to connect it to the abstract stage. This is where the intermediate stage plays an important role because it makes a bridge between the concrete and abstract stages and allows the student to see the essential parts of the concept that

remain constant as they move through the three stages. Some studies, however, do not include intermediate fading between the concrete and abstract stages (Goldstone & Son, 2005; Kaminski et al., 2008a, 2008b; De Bock et al., 2011; Trninic et al., 2020). Research on concreteness fading therefore diverges (Figure 2.3) and there is no clear consensus on what constitutes a 'correct' approach, although recent attempts have been made to further solidify it as a theory of instruction (Fyfe & Nathan, 2019). What is clear is that including only the concrete stage results in the lowest performance (Fyfe et al., 2015; Trory et al., 2018).

Figure 2.3

The Variety of Study Designs Falling Under the Category of Concreteness Fading

3 STAGES:
Physical: Concrete, Semiconcrete, Abstract (Fyfe et al., 2015; Trory et al., 2018; Kim, 2020)

Virtual: Concrete, Semiconcrete, Abstract (Arawjo et al., 2017; McNeil & Fyfe, 2012; Suh et al., 2021; Bronkhorst et al., 2021)

2 STAGES:
Virtual: Concrete, Abstract (Goldstone & Son 2005; Kaminski et al., 2008; De Bock et al., 2011; Braithwaite and Goldstone, 2013; Trninic et al. 2020)

2.3 Prior Work on Concreteness Fading

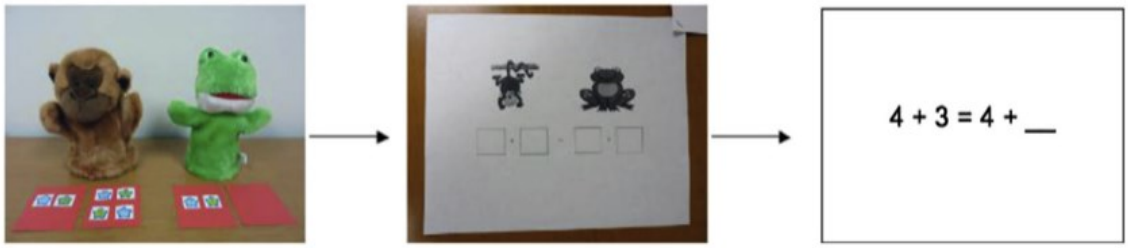
Concreteness fading has been investigated through several research programs. A prominent one is the work of Fyfe and colleagues. Fyfe, McNeil, and Borjas (2015), tested concreteness fading in the context of children's mathematical understanding. The participants were 63 children aged 7-9. The experiment included the following conditions:

- *Concrete only:* children received instruction with only concrete materials (frog and monkey puppets with stickers).
- *Abstract only:* children received instruction with abstract symbols of mathematical equations (4 + 3 = 4 + __).
- *Concrete to Abstract fading:* This is the 'concreteness fading' condition. Children received instruction in three stages – first with concrete materials, then with worksheets containing a 'faded' version of the concrete stage, and finally with the abstract mathematical problems (Figure 2.4).
- *Abstract to Concrete fading:* This condition is the reverse of the Concrete to Abstract fading condition. Children received instruction first with abstract problems, then the faded worksheet, and finally with concrete materials.

The participants completed an instruction phase and then a transfer test phase. The results show a main effect of instruction where children in the Concrete to Abstract (concreteness fading) condition performed better on transfer problems than children in the other three conditions. Moreover, the concrete-only condition produced the lowest learning compared to the other three conditions, and the concreteness fading condition yielded the best results. This pattern has also been found in other studies (Trory et al., 2018) but with some exceptions (Goldstone & Son, 2005; Kaminski et al., 2008a, 2008b).

Figure 2.4

Concreteness Fading Condition in Fyfe et al., 2015 Study



Note. From left to right, showing progression of instruction fading from concrete to semiconcrete to abstract.

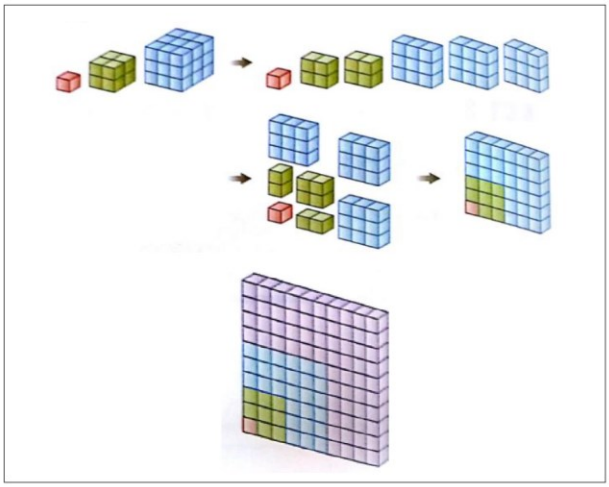
Adapted from "Benefits of "concreteness fading" for children's mathematics understanding" by Fyfe, E. R., McNeil, N. M., & Borjas, S. (2015). Learning and Instruction, 35, 104–120.

Kim (2020)'s study is one of the other few examples of concreteness fading done with three stages, with the concrete stage being in a physical environment. The lesson was taught to students in an all-female high school in South Korea (N = 35). The aim was to teach the abstract topic of summation of three finite sequences: $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$, $\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$ and $\sum_{k=1}^{n} k^3 = \left\{\frac{n(n+1)}{2}\right\}^2$. Instruction was first introduced using concrete magnetic blocks, then visual graphic representations, and finally the abstract algebraic notation. The study involved a single condition where a teacher presented this subject using the concreteness fading approach. The teacher guided students from the beginning where they used magnetic blocks to represent the sums above, until the end where they generalized the concepts using mathematical notation. Specifically, students were first given magnetic blocks which they could use to build their own models of the finite sums. If students struggled with the concrete material, they had access to worksheets

with hints containing graphical representations of how to assemble the blocks correctly (Figure 2.5).

Figure 2.5

Visual Representation Used in the Pictorial Stage of Teaching Finite Sums



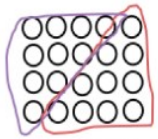
Note. From "Concreteness Fading Strategy: A Promising and Sustainable Instructional Model in Mathematics Classrooms" by Kim, H. (2020). *Sustainability*, 12(6), 2211.

Students then extended their understanding of the patterns in these visual representations by adding the mathematical symbols themselves (numbers, summation sign, etc.) (Figure 2.6). They were encouraged to notice the connections between each representation (concrete manipulatives, visual representations, and abstract mathematical notation). Afterwards, students presented their knowledge to the class demonstrating their understanding of the links between these concepts. Overall, using the concreteness fading approach during the lesson was shown to be useful during the student presentations.

This case study illustrated how to teach abstract mathematical material using concreteness fading in a classroom, providing insight for teachers on how to support students' conceptual understanding in these abstract mathematical domains.

Figure 2.6

Progression from Concrete to Abstract Instruction of a Finite Sum

Concreteness Fading Phases	Concrete Phase	Connecting Phase	Pictorial Phase	Connecting Phase	Abstract Phase
$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$	Step-by-step approach. Representation of 1 + 2 + 3 using magnetic blocks first. 	Support in representing 1 + 2 + 3 + 4 both using a pair of blocks of 1 + 2 + 3 + 4 and implicitly expanding it into an abstract representation. 	Asking students how to represent $1 + 2 + 3 + \ldots + n$ using a diagram. One student represents the exemplary diagram: 4+3+2+1  1+2+3+4	Supporting students in expanding this diagram to generalize it to $1 + 2 + 3 + \ldots + n$ by asking facilitating question, "How can we represent the finite sums of natural numbers using this diagram? Can you generalize how many dots are in the diagram?"	Students use two ways to represent the algebraic forms of $\sum_{k=1}^{n} k$: (a) $n \times (n+1) \div 2$ (b) $\frac{n}{2} \times (n+1)$ Ms. K summarizes that these two algebraic forms are the same as $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.
Instructional support strategies	Ms. K focuses on sense-making of each concept by supporting students in operating the magnetic blocks (i.e., concrete objectives and physical activities) with step-by-step questions.	Ms. K supports her students in representing simpler concepts with magnetic blocks, and also in interpreting their activities with manipulatives in mathematically meaningful ways.	Ms. K focuses on concept development and meaningful support by providing visual representations, especially when students are struggling with making connections between concrete activities and abstract representations.	Ms. K guides students in deleting superficial features from manipulatives or visual presentations. She highlights core structures of mathematical concepts.	Ms. K fosters students in using abstract representations of the concept by supporting generalizations and core abstract concepts. She also supports students in generating different algebraic forms of a concept from their concrete and visual representations.

Note: The progression from concrete (left) to abstract (right) instruction for a finite sum. Note that the pictorial phase, contains visual representations that have both less perceptional details (colours are removed, simple circles are used) and elements of the abstract stage are also introduced (namely numbers and addition signs). Moving from the pictorial phase to the abstract phase, all graphical elements are removed and only the mathematical symbols remain.

Adapted from "Concreteness Fading Strategy: A Promising and Sustainable Instructional Model in Mathematics Classrooms" by Kim, H. (2020). *Sustainability*, *12*(6), 2211.

Trory et al. (2018) also involved the concreteness fading paradigm with three stages. Their aim was to teach computing concepts such as network structure and internet routing tables to young students. Participants were 59 primary school children (aged 9-10). The study used a between-groups design with four conditions: abstract, concrete, concreteness fading, and concreteness introduction (Table 2.1).

Table 2.1

Organization of the Different Stages by Condition in Trory et al., 2018

	Stage 1	**Stage 2**	**Stage 3**
Abstract-only	Abstract	Abstract	Abstract
Concrete-only	Concrete	Concrete	Concrete
Concreteness Fading	Concrete	Intermediate	Abstract
Concreteness Introduction	Abstract	Intermediate	Concrete

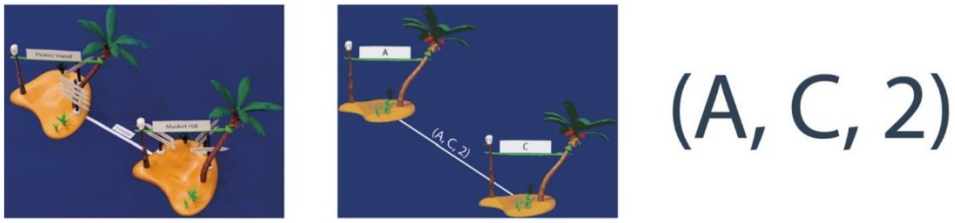
In the concreteness fading condition, concrete concepts were introduced in a physical environment, meaning that objects were created using toys. To illustrate, for networks, the concrete stage was an analogy of a group of islands with a parrot travelling between islands and carrying messages to their inhabitants. The islands were intended to represent nodes in a network, the path of the parrot the edges between the nodes, and the parrot itself the notion of a data packet. For the second stage, the same ideas were now represented graphically, and elements of the final abstract notation were also introduced. This facilitates the movement to the third stage, which consisted solely of the abstract notation as seen in Figure 2.7.

The results showed a significant difference in posttest scores between conditions, with the concreteness fading condition leading to highest learning. There was a marginally significant difference between the concreteness fading condition and the concrete

condition, however, in contrast to other studies cited above, no significant difference between the concreteness fading condition and the abstract condition.

Figure 2.7

Concreteness Fading Stages for Teaching Network Structure and Node Edges

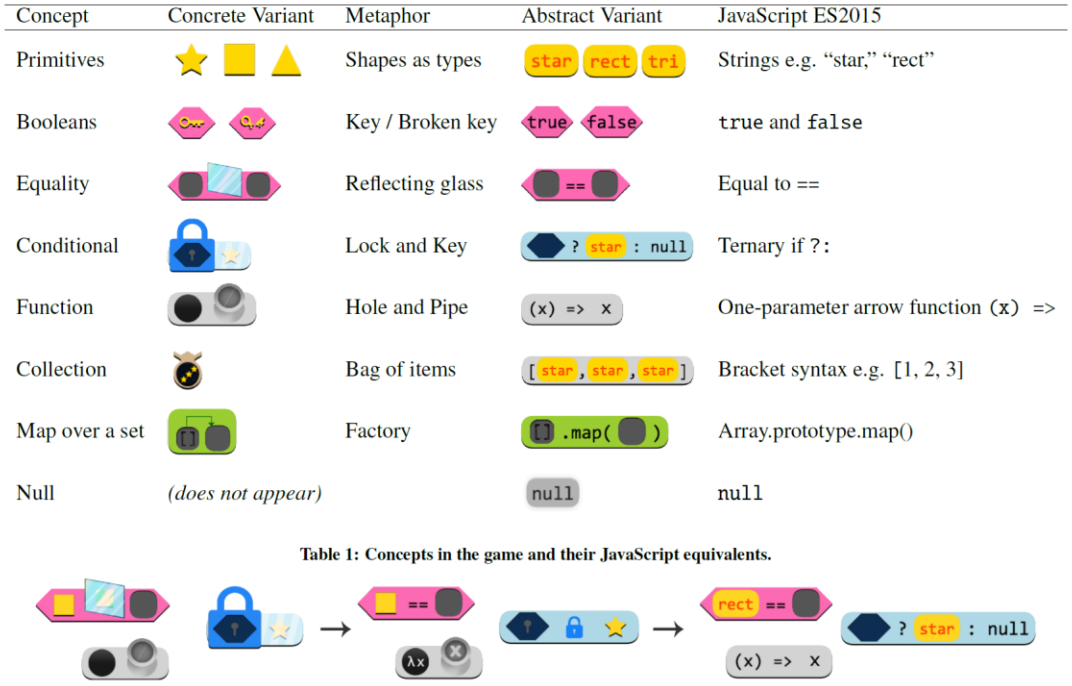


Note. From "Designing for concreteness fading in primary computing" by Trory, A., Howland, K., & Good, J. (2018). *Proceedings of the 17th ACM Conference on Interaction Design and Children*, 278–288.

Arawjo et al. (2017)'s work is one of the few examples of concreteness fading entirely in a virtual environment and in the domain of programming. This study, however, did not apply concreteness fading to teach programming as a lesson, but rather as a game (contrary to other work on concreteness fading discussed in this chapter). In this game called *Reduct*, programming concepts were first introduced through concrete representations related to real-world objects. The conjecture was that these would be easier to grasp compared to seeing the actual code first. For example, a mirror (reflecting glass) was used to represent the concept of equality (Figure 2.8).

Figure 2.8

Concrete Representations of Abstract JavaScript Code



Concept	Concrete Variant	Metaphor	Abstract Variant	JavaScript ES2015
Primitives		Shapes as types	star rect tri	Strings e.g. "star," "rect"
Booleans		Key / Broken key	true false	`true` and `false`
Equality		Reflecting glass	==	Equal to ==
Conditional		Lock and Key	? star : null	Ternary if `?:`
Function		Hole and Pipe	(x) => x	One-parameter arrow function `(x) =>`
Collection		Bag of items	[star, star, star]	Bracket syntax e.g. [1, 2, 3]
Map over a set		Factory	[].map( )	Array.prototype.map()
Null	*(does not appear)*		null	`null`

Table 1: Concepts in the game and their JavaScript equivalents.

Figure 5: From left to right: Pieces of the game at progressive stages of concreteness fading, starting from the most concrete and ending at the most abstract.

Note. The bottom of the figure contains a specific progression of a JavaScript concept from concrete to semiconcrete to abstract (left to right).
From Arawjo, I., Wang, C.-Y., Myers, A. C., Andersen, E., & Guimbretière, F. (2017). Teaching Programming with Gamified Semantics. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 4911–4923.

The participants were 24 undergraduate students with no experience in programming. They were asked to play the game from the beginning (all 72 levels of it), with no time constraints. The study had a between-subject design. Half of the participants played the game with a concreteness fading condition that had 3 stages, while the other half played the game with the "fully faded" condition (this is essentially the abstract-only condition, as it consists of mainly just code). The results showed no significant effect of condition on the various test components (recognition, recall, near-transfer, and total

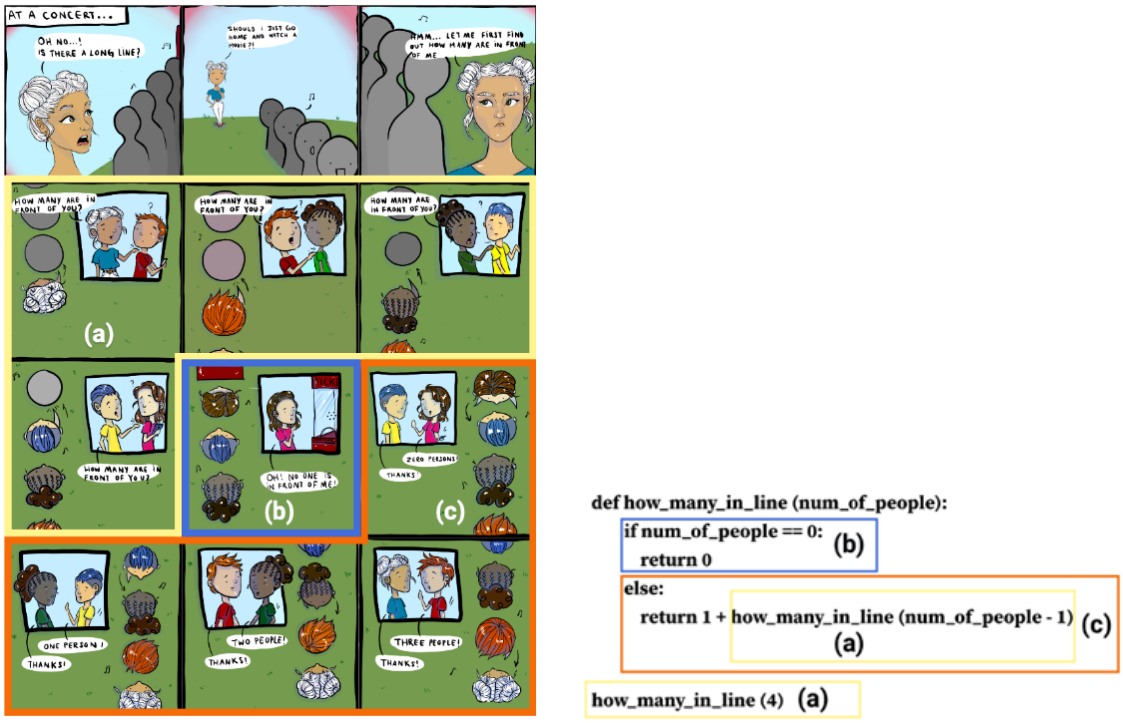
scores). One issue, however, is that the game did not include instruction on the target concepts. Players were left to discover the rules on their own, but as discussed in Section 2.2, the approach of concreteness fading works best when accompanied with guidance on the mapping and relationships between the representations. Another concern is that the representations in the various concreteness levels were confusing, based on our analysis and from interacting with the game. For example, it is not self-evident that the mirror represents equality, or that a lock and a key represent a conditional statement. A good implementation of the concreteness fading approach includes clear links between representations as well as guidance from the teacher regarding said links. Had these links been explained even slightly, then the representations would have made more sense and have been easier to grasp from the start.

Research on concreteness fading is still an emerging field. Recently, Suh et al. (2021) also implemented concreteness fading to teach programming concepts. They proposed a method called *coding strip:* a comic strip showing both the cartoons and the code so that links can be made between them (Figure 2.9). They used this method to teach an undergraduate first-year computer science course ($N = 49$) and surveyed the students to identify the benefits and drawbacks of this approach. Their results showed that most students found it beneficial to learn programming through comic strips.

Although Suh et al. (2021) applied concreteness fading to teach programming concepts, their research differs from the current thesis because comics are more static than our approach. Specifically, in this thesis, the programming concept is presented dynamically using an animation. Moreover, Suh et al. (2021) do not yet have experimental data measuring the learning effects of their approach.

Figure 2.9

Concept of 'Recursion' in Programming, Using the Coding Strip Approach



Note. Adapted from "Coding Strip: A Pedagogical Tool for Teaching and Learning Programming Concepts through Comics" by Suh, S., Lee, M., Xia, G., & Law, E. (2020). *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 1–10.

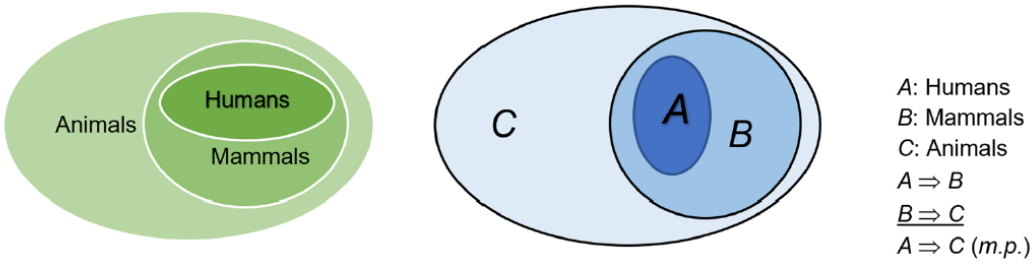
Bronkhorst et al. (2021) aimed to teach logical reasoning through different modes of representation (enactive, iconic, symbolic) by fading between them using the concreteness fading method. The study used a single condition and qualitative analysis of the results. The participants were seven 'non-science' grade 12 students (last year of high school) in the Netherlands. Students received ten 50-minute lessons on the topic of logical reasoning, and the authors then analyzed the video recordings of the group of students during these lessons. The first two lessons explored logical reasoning in contextually grounded concrete scenarios using common words, such as in newspaper articles. This was

the ‘enactive mode’, as it used verbal descriptions of realistic real-world scenarios. The following lessons consisted of working with visual representations of these concrete logical scenarios. Attention was paid to the links between representations. This visual stage was their ‘iconic mode’ and had two steps: first visualising the concrete representations using letter symbols and then incorporating them into Euler and Venn diagrams. Thus, if students were able to establish the link between the enactive concrete stage and the iconic (Venn diagram) stage, this demonstrates that they were able to use generic letter symbols like A, B, P, Q and apply them to the concrete word-examples by using them in a visual representation like a Venn diagram (Figure 2.10). In the final ‘symbolic mode’, students discovered the rules of logic such as ‘modus ponens’ (A => B, A therefore B) using formal notation with logical symbols ($\wedge$, $\vee$, $\Rightarrow$, and $\neg$) (Bronkhorst et al., 2021) (Figure 2.10). Throughout the lessons, students were encouraged to discover these links by themselves, with guidance from the teacher pointing out the relationships between the representations afterwards.

Based on their analysis, the authors concluded that the concreteness fading model was useful. The hardest links for students to make were between the iconic and symbolic modes, and students often went back to visual representations when moving from concrete to symbolic via iconic.

Figure 2.10

Concreteness Fading of the Concept of Logical Reasoning



Note. Fading from concrete (left) to semiconcrete (middle) to abstract (right) of the concept of logical reasoning, with the specific example of the following syllogism: "All humans are mammals. All mammals are animals. (So) All humans are animals."
From "Student Development in Logical Reasoning: Results of an Intervention Guiding Students Through Different Modes of Visual and Formal Representation" by Bronkhorst, H., Roorda, G., Suhre, C., & Goedhart, M. (2021). *Canadian Journal of Science, Mathematics and Technology Education*, *21*(2), 378–399.

In their study, Goldstone and Son (2005) also presented their material in a virtual environment, but the lesson only had two stages. Although their approach does not exactly follow the concreteness fading steps outlined in Section 2.2, it is nevertheless notable research as it is among the earlier studies investigating this approach. Their goal was to teach the concept of 'competitive specialization', which is an important notion in cognitive science, where *"the parts of a system can organize themselves without the help of a leader or centralized plan"* (Goldstone & Son, 2005), similar to adaptive behavior. This behavior can be found in neurons as well as animals consuming resources in a given territory. If the agent closest to a food source moves faster than the rest, they will have access to it before the others, hence the competition. If all agents move at the same time, they would get to the food source all at once. So other agents should move at different speed to have access to other sources. In this study, the researchers used the example of ants foraging food, presented in a virtual environment. Participants could interact with the simulations and

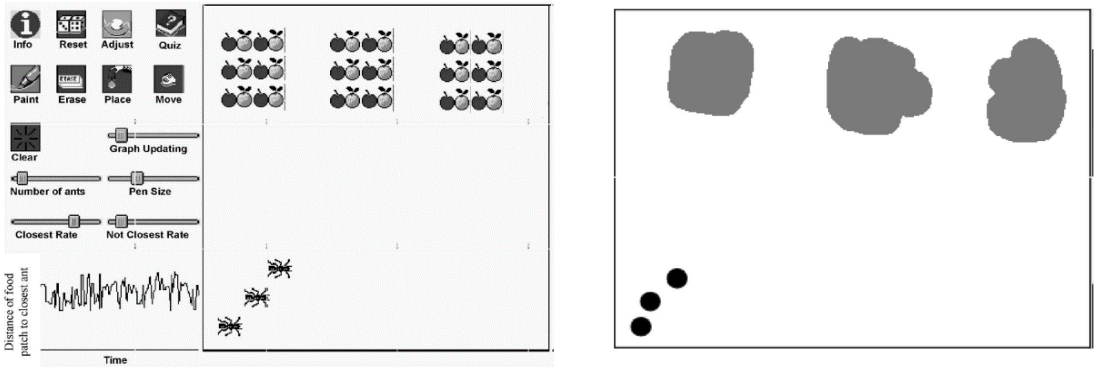
change certain features such as the movement speed of the ants. This allowed them to explore which factors influence how the resources will be allocated.

The participants were 84 undergraduate students, split evenly among the four conditions of the experiment: concrete, abstract (idealized), idealized-then-concrete (concreteness introduction), and concrete-then-idealized (concreteness fading). The last two conditions represent the fading paradigm but with only two stages – concrete and idealized. The concrete stage was the simulation of ants and food (in this case apples) represented graphically. In the idealized stage, small black dots were used to represent ants, and larger green dots to represent food (shown in grey in Figure 2.11). In the concreteness fading condition, concrete versions of ants and food were used for the first 10 minutes. After that, participants were informed that *"we are now changing the appearances of the food and ants, but they still behave just as they did before"* (Goldstone & Son, 2005, p.81). From that point, the idealized versions were used.

The results show that performance was highest in the concreteness fading condition and lowest in the idealized-only condition, with a significant difference in transfer performance.

Figure 2.11

Concrete vs. Idealized Representation of Ants Foraging Food

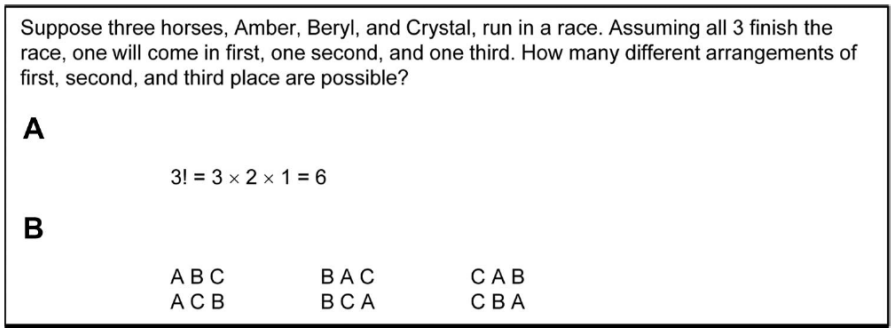


Note. Left showing concrete representation of ants foraging food. Right showing idealized version with black dots representing ants, and larger grey dots food.
Adapted from Goldstone, R. L., & Son, J. Y. (2005). The Transfer of Scientific Principles Using Concrete and Idealized Simulations. *Journal of the Learning Sciences*, *14*(1), 69–110.

Braithwaite and Goldstone (2013) used the concreteness fading approach to teach combinatorics in mathematics. Combinatorics is a domain focused on 'counting', or more formally, calculating the number of possible outcomes of separate events. In this study, the combinatorics equations were first represented in a concrete stage using outcome listing. This is a way of listing all the possible outcomes of a situation (Figure 2.12 B). For the abstract stage, the combinatorics calculations were formally represented using their (abstract) mathematical notation. There are therefore two representations: listing (concrete) and calculations (abstract).

Figure 2.12

Abstract vs. Concrete Representation of a Combinatorics Problem



Note. A combinatorics problem showing a solution using abstract numerical calculation (A) and one using a concrete, outcome listing, representation (B).
From "Integrating formal and grounded representations in combinatorics learning" by Braithwaite, D. W., & Goldstone, R. L. (2013). *Journal of Educational Psychology*, *105*(3), 666–682.

The study consisted of two experiments with outcome measures corresponding to pretest to posttest learning gains. Experiment 1 compared 4 different types of instructions:

- Concrete-only (listing only)

- Abstract-only (numerical calculation only (what they call pure formalism))

- List fading (listing then calculations (this is concreteness fading))

- Formalism-first (calculations then listing)

Participants were 111 undergraduate students in the USA. In each condition, participants were presented with four problems, and before moving to the next problem saw a sample solution to the current problem. What differed between the conditions is how the participants were instructed to solve the problem, and the sample solutions that they were shown. For the abstract-only condition, all problems had to be solved using calculations only, and the sample solutions also showed calculation-only solutions. For the concreteness fading condition, the first problem was a listing-based solution, then the

second and third problem were a mix of both a listing-based and a calculation-based solution. The last problem was a calculation-based solution.

Results showed that list fading (concreteness fading) and pure formalism (abstract-only) lead to similar transfer performance, and concreteness fading had higher performance than formalism-first and listing only (concrete only). The authors concluded that combining concrete and abstract representations can be an effective way of teaching combinatorics, but concreteness fading might not necessarily offer an advantage over purely abstract approaches that use calculations only. They do however acknowledge that their results could be due to implementation weaknesses, correctly noting that instruction using grounded representations is often affected by the details of how it is implemented. In this experiment, the grounded (concrete) outcome listing representation did not perceptually resemble the formal (abstract) calculations representation, making it difficult to transition from one to the other in the concreteness fading condition. Indeed, large jumps between representations are not recommended (Fyfe et al., 2014). In addition, the relationships and relational correspondence between both stages were not described, as is encouraged by the theory of concreteness fading (see Section 2.2).

Braithwaite and Goldstone (2013) devised the second experiment with aims of rectifying these weaknesses. Their goal was to investigate whether the results found in experiment 1, namely that the concreteness fading condition was similar to the abstract-only condition, would be replicated, but with more extensive transfer problems. They also implemented additional instruction, as audio rather than on-screen text, to help participants understand the connections between both the concrete and abstract representations

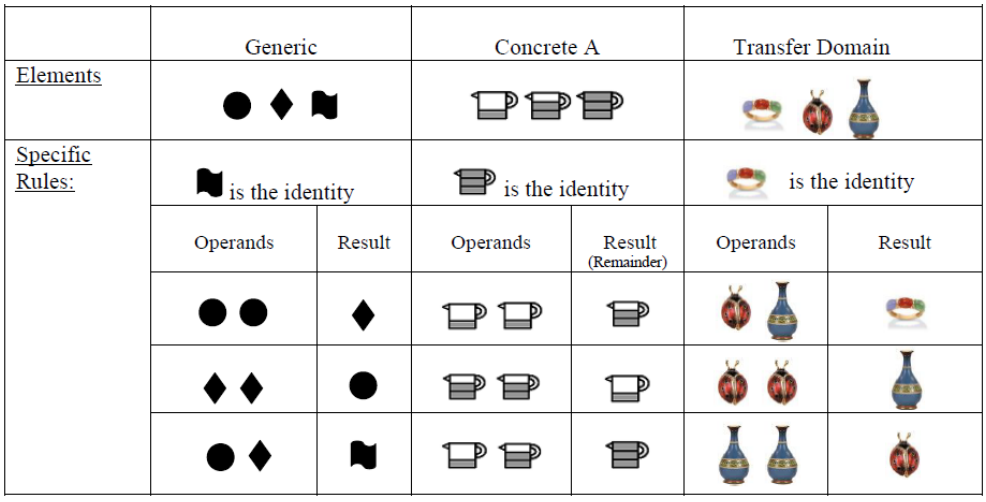
(outcome listing and formal calculations respectively). The participants were 73 undergraduate students from the same American university.

Despite these changes, results from experiment 2 mirrored the findings of experiment 1 – there was no significant difference between the concreteness fading and abstract-only groups in transfer performance.

Some studies have challenged the approach of concreteness fading. Kaminski et al. (2008a, 2008b)'s work implemented a lesson on the concept of a commutative mathematical group of order three, to 40 undergraduate students. Participants who learned only the generic (abstract) representations performed better in a transfer domain (with a task involving 'real-world' objects) than participants who learned both the concrete then generic instantiations (Figure 2.13).

Figure 2.13

Generic (Abstract) and Concrete Representations of a Mathematical Group of Three



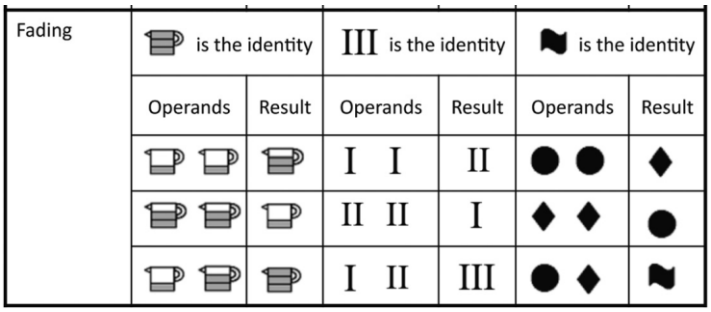
Note. From "Supporting Online Material for: The Advantage of Abstract Examples in Learning Math" by Kaminski, J. A., Sloutsky, V. M., & Heckler, A. F. (2008b). *Science*, *320*(5875), 454–455.

De Bock et al. (2011) and Trninic et al. (2020) both aimed to replicate Kaminski et al. (2008a, 2008b)'s results. De Bock et al. (2011) (N = 105 undergraduate students) found participants who received abstract instruction performed better on abstract transfer tasks than participants who received concrete instruction. However, participants who received concrete instruction performed better on a concrete transfer task than participants who received abstract instruction. Trninic et al. (2020) (N = 76 undergraduate students) found that on abstract transfer tasks, there was no significant advantage of learning the material through concrete or abstract representations.

The above studies only included two stages - from concrete to abstract, without an intermediate stage to link them together. This therefore does not entirely follow the guidelines of a proper concreteness fading approach outlined in Section 2.2. To improve this, McNeil and Fyfe (2012) conducted a study following Kaminski et al. (2008a, 2008b) and De Bock et al. (2011)'s results, using identical materials, but included an intermediate stage between the concrete and abstract stages of instruction (Figure 2.14). They found that instruction with fading between three stages performed best on transfer tasks.

Figure 2.14

Concrete, Faded, and Generic (abstract) Representations of a Group of Order Three



Note the addition of the middle phase with Roman numerals which was not included in the original Kaminski et al., 2008a work.

From "'Concreteness fading' promotes transfer of mathematical knowledge" by McNeil, N. M., & Fyfe, E. R. (2012). *Learning and Instruction*, *22*(6), 440–448.

2.4 Summary

As described in this chapter, although there is progress in the field of concreteness fading, studies in this area are not all in agreement with each other in terms of their findings as well as in the approach they use. As far as the latter, some studies use three stages for concreteness fading, others only include two. Some have physical manipulatives in their initial concrete phase, others are entirely conducted in a virtual environment. Some studies have children as participants, others have older students in high school or university. Most studies are in the domain of mathematics, but there are also studies in the programming field. Some studies find concreteness fading to be a beneficial pedagogical approach leading to higher learning gains and transfer, while others do not find significant differences between abstract-only, concrete-only, and concreteness fading conditions. Overall, although the concreteness fading paradigm provides several important instructions on how to move from concrete to abstract representations with links between each stage, additional research is required to find optimal ways of teaching these difficult abstract concepts using this method. The present thesis aims to add to the existing body of research on this topic.

Chapter 3: Design of the Instructional Materials

The lesson in the present study covered the concept of a for-loop. A for-loop iterates over a sequence of elements, repeating instructions inside the loop body. Figure 3.1 shows an example of a for-loop in the Python programming language. Here, the sequence is a list of numbers (line 1, Figure 3.1). For each iteration of the for-loop, the current element in the sequence is assigned to the variable 'integer' (line 5; this variable can be called anything). The flow of execution then enters the body of the for-loop, i.e., the indented part (lines 6-7) and implements the instructions there. In this case there is an if-statement (line 6) checking if the current element is greater than 15. If it is, then the variable 'count' is updated by one. The program will repeat the loop until it has gone through each element in the sequence one-by-one. For-loops are one of the fundamental building blocks of computer programming. Other building blocks, such as variables and conditional statements (e.g., if-statements), are more basic, and so loops are typically taught after the basics. Students often struggle with loops (Robins & Rountree, 2003) making it an appropriate choice for this research, as work is needed to identify effective teaching methods.

Figure 3.1

Example of a For-Loop in Python

```
1 numbers = [20, 5, 37, 109, 12, 8]
2
3 count = 0
4
5 for integer in numbers:
6     if integer > 15:
7         count = count + 1
8
9 print(count)
```

There are different types of loops, but while-loops and for-loops are the most common. In contrast to a for-loop, while-loops iterate until a condition is no longer true. It is not clear if one loop is more difficult to learn than the other, but anecdotally based on teaching experiences at Carleton, students struggle with for-loops in particular, even for simple programs. This may be because some functionalities are implicit (e.g., that the current list element is assigned to a variable, called integer in Figure 3.1). There are certainly ways to make while-loops more complex, for instance by using break statements to stop the loop under certain conditions (there would then be two conditions to keep track of, the one after the while and the one after the if). However, we wanted to focus on the base mechanics of the loop by teaching the main concept of a loop without additional details that could confuse a student. We therefore chose to use for-loops.

We decided to keep the loop fairly basic because once the core of the concept is understood, it can be applied to more complex situations. However, we did include one conditional statement in the body of the loop because most loops are built to check a certain condition, and this builds on the students' current knowledge of conditionals.

We begin by describing considerations related to the design of the concreteness fading materials. Since the concreteness fading approach relies on moving from a concrete to an abstract representation of the target concept, we needed to find an appropriate concrete representation of a for-loop. The first step for doing so involves identifying important components of a loop and developing a correct model of how it operates.

3.1 Loop Components, Mental Models

The important components of a loop include a sequence (e.g., string, list) that the program iterates over, and an index variable that references each element of the sequence once per iteration of the loop – the sequence elements get assigned to this variable. The index is external from the sequence, and can be mentally operationalized like an arrow pointing at a specific element of the sequence, without the arrow being visible in the program. For instance, in Figure 3.1, the index variable is 'integer' and the sequence is the list 'numbers'. The index variable is a difficult concept because the variable name is arbitrary, and the assignment of each sequence element to it happens implicitly.

Although the various parts of a loop can be complex to grasp, having only knowledge of loop components is not enough to fully understand how they work - students must also be able to simulate them via mental models which will help them learn programming. Mental models are *"psychological representations of real, hypothetical, or imaginary situations"* (Johnson-Laird et al., 1998). In other words, a mental model is an internal representation or simulation of how something works in external reality. A standard programming-related model is the notional machine, *"a crisp, human-friendly abstraction that explains how programs execute in a given language [...] ie. a model of computation"* (Krishnamurthi & Fisler, 2019). Hence, a notional machine is a model of how the computer executes a program. Novice programmers often construct notional machine models that are inconsistent with the actual behavior of the program (Sorva, 2013). This can be aggravated by the fact that the semantics in natural language and programming may be different. For instance, the term 'for' in a for-loop does not intuitively suggest some sort of repetition (loop).

Below is one possible mental model of how a for-loop operates:

Step 1 – Look at the current element in the sequence. At the start, the current element is the first unprocessed sequence element – initially at position 0 then 1, 2, and so on.

Step 2 – Assign it to the index variable.

Step 3 – Enter the loop body.

Step 4 – Execute the instructions inside the loop.

Step 5 – Go back to the top of the loop.

Step 6 – Repeat Steps 1 – 5 until all the elements in the list have been examined.

This model remains the same regardless of the specific details of the loop. The key features of this model should be demonstrated/instantiated in the concrete representation – how to do so was informed by work on analogy that we next describe.

3.2 Analogy and Related Frameworks

One way to do identify key features of a mental model that need to be instantiated in a concrete representation is to make an analogy between the concrete and the abstract stages. An analogy is a comparison between two concepts and we use this comparison to learn about new, unknown concepts. Specifically, an analogy is *"the process of understanding a novel situation in terms of one that is already familiar"* (Gentner & Holyoak, 1997). The familiar situation serves as the *base* or *source* that provides a model to make inferences and understand the unfamiliar situation, also called the *target* (Gentner & Holyoak, 1997). In our case, the target is the unfamiliar concept of a for-loop presented as Python code in a computer program. The source will be a familiar situation that is grounded in real-world experience. This is the concrete stage of the concreteness fading approach. The challenge is to find the most appropriate example of a source situation that

represents the concept of a for-loop, but that students can relate to more easily than plain code.

The Structure Mapping Engine (SME) can help answer the question of how to link the source and target domains. SME is a theoretical framework that proposes an analogy is conceptualized as a mapping between the familiar (source) domain and the unfamiliar (target) domain (Gentner, 1983). Thus, an analogy between two domains means that there is a relational structure in one domain that can be applied in the other domain - there is 'structural alignment' between the domains (Gentner & Holyoak, 1997). This alignment or mapping allows for transfer of knowledge from one situation to another. The mapping step requires identifying one-to-one correspondences between the two domains in question (Gick & Holyoak, 1983), where each element in a representation is paired with no more than one element in the other representation (Gentner et al., 1997).

We relied on the analogy research to explore which real-world concept(s) has a relational structure similar to that of a for-loop. We also relied on two additional frameworks, namely the Cognitive Alignment Framework and Epistemic Fidelity.

The Cognitive Alignment Framework suggests that *"the more precisely that physical materials and learning activities are aligned with the desired mental representation, the more likely students are to acquire that representation"* (Laski & Siegler, 2014). Moreover, when physical materials are closely aligned to the target mental representation, analogical transfer between the physical and abstract representation is increased (Laski & Siegler, 2014). Since we want to create a lesson that will help students gain more accurate mental models of for-loops and ultimately help them understand the concept better, we paid attention to aligning the concrete representation to the abstract

mental model, essentially creating an analogy between both. Another relevant principle is epistemic fidelity. Stacey et al. (2001) defines epistemic fidelity of instructional materials as *"a measure of the quality of analogical mapping [quality of the links] between the features of the material and the target knowledge domain"*. We strived to create a lesson with high epistemic fidelity, meaning that the relationships between the features of both representations of a for-loop (the concrete representation, and the abstract code which is the target domain) were strong. Clear links between representations is an important component of concreteness fading as explored in Chapter 2.

3.3 Deciding on the Correct Design for the Concrete Phase

The work presented above provided the foundation to help us find a concrete representation of a for-loop that was based on a familiar real-world domain and design the instructional materials for it. The objective was to identify concrete constructs and situations that are repetitive and involve a sequence.

We had several potential candidates of real-world scenarios that could represent the idea of going through a sequence one element at a time and repeating a series of actions for each element. Some examples include depositing money into a bank account, selecting a ball of a certain colour from a sequence of balls, a lineup at a coffee shop, a conveyor belt with boxes, a row of houses on a street, counting rocks with a specific feature (for instance smooth), sorting candies by shape, etc. The scenario we settled on was going through a sequence of books on a bookshelf and doing certain operations for each book.

We needed a concrete way to represent the index variable. Recall this variable references a single element of the sequence that the loop is currently operating on. Thus, the program can only "see" one of the sequence elements at a time. To make this concrete,

we chose to use a magnifying glass. If a magnifying glass is needed to see an item, the item will only be visible when looking at it with the magnifying glass. Otherwise, it will not be visible because it is too small to see. The magnifying glass hence reveals each element when it is its turn (in other words, when the index refers to the current element).

In a loop, the flow of execution involves both horizontal and vertical movement: (1) elements are accessed *across* the sequence (horizontal), (2) for each element, the execution jumps inside the loop body, and here the computer executes the program from the top to the bottom of the body of the loop (vertical movement). The visualization of this flow of execution does not need to resemble an actual loop (cycle) as this is not an entirely accurate representation. In other words, we might intuitively think of a loop as a cycle and look for an example of that in the real-world, for instance a bicycle wheel. However, the visual representation of a for-loop (as code) does not resemble a wheel, even though its algorithm performs a cyclical loop-like motion.

Simulating the computer movement can help students understand the notional machine model more correctly, because the concept they need to learn is that the computer is doing the movements and not the student. We used a robot to represent the computer. We chose a robot rather than a human arm to help students distance themselves from the idea that they are the ones going through the sequence, when in reality, it is the computer executing the program that is doing it. This was designed to help them embody the computer.

We incorporated these design ideas into the concrete representation of a for-loop and now provide a high-level description of the final version. A robot arm moves through a sequence of books on a bookshelf. Each book contains a label consisting of a single letter

written on it. The robot can only see things from up close, so it needs to use a magnifying glass to see the label because it is too small otherwise. For each book it looks at one-by-one, it moves to its worktable below the bookshelf, and follows the instructions on the notebook. The instructions tell the robot to update its count of the number of books with the letter E each time it comes across a book with the letter E. This is important: the robot cannot remember instructions like humans, so it forgets them right after it follows them – so it has to come back to this worktable each time it sees a new book on the shelf. The idea of repeating the action of looking at a book on the bookshelf, and then coming back to the worktable to read the instructions, and then returning to the bookshelf and looking at the next book in the sequence simulates the operation of a for-loop.

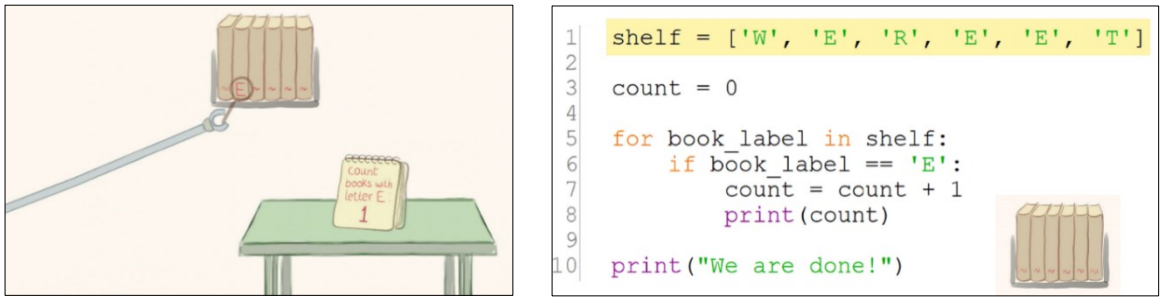
3.4 Structural Mapping between Domains

We can verify that our analogy is strong by checking the mapping between our source and target domains, following the process described in the Structure Mapping Engine (Gentner, 1983). The target domain is the code representing a for-loop going through elements of a list. A strong analogy is one where the *relationships* between the parts in the source and the target situations are similar to each other but the literal similarity of the parts themselves is not important (Gentner & Holyoak, 1997). Essentially, if our source representation has concrete features that are abstracted during the mapping from source to target, then we are dealing with an analogical comparison. So in our robot example, the specific features of the robot are not what get mapped onto our target (e.g., a robot arm going through books on a bookshelf is not physically the same thing as a computer executing a program). Instead, what matters is the relationships between the components of the concept: how the robot moves through the bookshelf book by book is

analogous with how the computer moves through the list element by element (Figure 3.2). Thus, there is structural alignment between both domains. Here, the focus is on how the relationships between the constituents in each domain overlap rather than how many of the literal appearance features overlap. Table 3.1 illustrates this by providing a structural mapping between the various components and connections in our source and target domains. Having a strong relationship between both domains is key. For instance, if the instructions state that "the computer executing the code" is like "the robot following its instructions" this makes explicit a mapping from the source domain (the robot) to the target domain being taught (the computer) (Gentner, 1983). Hence, this helps students make a connection from existing knowledge to the unknown concept (Table 3.1).

Figure 3.2

Relationship Between Components of the Concept



Note: The robot (left) is looking at book labels on the bookshelf one at a time, in a similar way as the computer (right) that is going through the elements of the 'shelf' variable using the book_label variable to refer to each element one-by-one. Yet, the robot arm and the computer look nothing alike physically.

Table 3.1

Structural Mapping Between our Source and Target Domains

Source domain (robot going through books on a shelf)	**Target domain** (for-loop going through elements of a list)
Robot (going through the books on the shelf)	**Computer** (executing the program)
Bookshelf with books on it. Each book has a label consisting of a single letter	**Shelf variable** that is a list, containing string elements representing the labels on the books in the shelf
Magnifying glass symbolizing that the robot can only see the current letter	Computer only sees the elements of the list one at a time
The **book label** on each book is revealed only once the magnifying glass looks at the current book	**Index variable (book_label)** keeps track of current element in the list (each element is a letter representing a book label)
Worktable is in a location separate from the bookshelf	**Body of the for-loop** is distinctly indented in the code to separate it from the sequence
Robot going to notebook on the worktable	Computer entering body of the for-loop
Instructions on the notebook that need to be checked for each book on the shelf	**Conditional if-statement** that needs to be checked for each element in the list
Count is updated and displayed on the notebook	**Count** is kept track of and printed by the program

3.5 Presenting the Concrete Representation through Animation

Once the concrete representation was chosen, we needed to decide how to present it. For-loops are a dynamic process because program execution jumps around (both horizontally and vertically as described above) and so a static image of the chosen concrete scene does not align well with the mental model of how a loop works. Therefore, an animation better portrays the dynamic nature of a loop in a program as compared to a static description (Esponda-Argüero, 2008).

We followed and adapted several general guidelines extracted from literature on how to design appropriate concrete instructional materials that correspond to animations. These included:

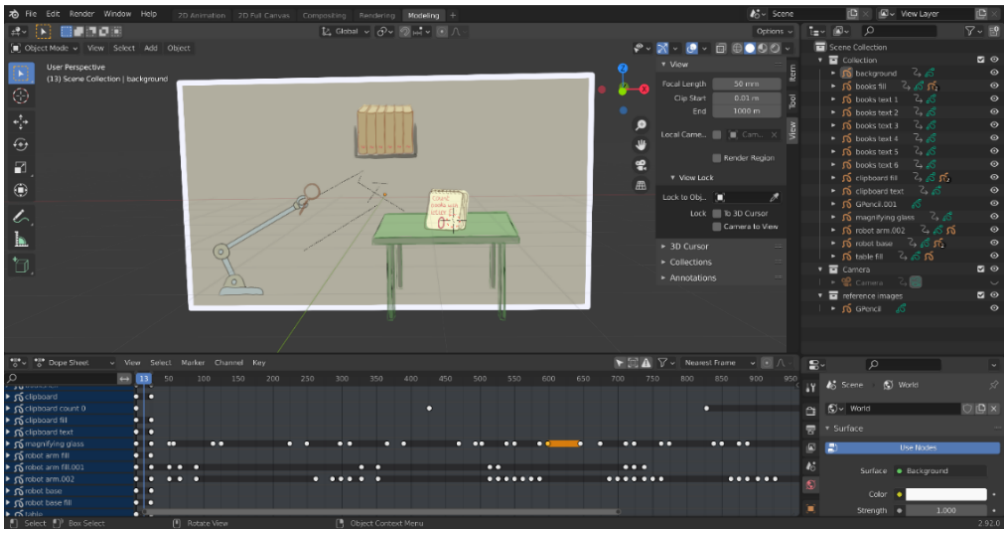
- Avoid concrete representations that have distracting irrelevant features (Laski et al., 2015; Al-Sakkaf et al., 2019). Removing unnecessary details reduces the salience of the concrete material and helps the student to instead focus on the structural information, which also facilitates knowledge transfer (Fyfe et al., 2014). Our animations are therefore simple, with muted colours, and only contain essential components, which is why the scene only has very few objects present in it.
- Along these lines, it is important to not to overload the working memory of the student (Al-Sakkaf et al., 2019), particularly considering that participants may have never been exposed to programming before. A simple animation of a concrete representation of the model should help reduce cognitive load compared to a more flashy animation.
- Following the framework of epistemic fidelity, a student needs to have the correct mental model of a for-loop, and if the visualization correctly corresponds to this model, then learning will occur (Esponda-Argüero, 2008). A successful animation will show a concrete model of a for-loop, and seeing that model will help the student recall the core procedures involved in loop mechanics, for when they move to the abstract stage.

These guidelines are also in line with the key factors of a proper concreteness fading approach, which were discussed in Chapter 2.

We created three initial animations. The first two animations show a robot arm going through book on a bookshelf with elements of code present, and the third animation code-traces a specific program (explained in more detail in Section 3.7.1). The software used to create them was Blender[1]. Blender is an open-source software used to create both 3D and 2D models, animations, and simulations. It is extremely powerful, but this makes its learning curve difficult and so significant time was invested by the thesis author to learn the basics of the software and interface, as well as create the actual animations. Once the software is mastered, it allows for a lot of flexibility with all its features. Figure 3.3 shows a screenshot of Blender's interface. In the middle is the scene we are creating. At the bottom we can see the keyframes for the animation. On the right we have our collection that contains all the objects in our scene.

Figure 3.3

Blender Interface Showing the Creation of the Animation Video



[1] https://www.blender.org/

3.6 Pilot: Testing the Analogy and Design of the Material

We ran a pilot study with four participants to obtain feedback on the design of the animations. The participants either did not have any prior programming experience, or at most had taken one programming course. Each session lasted approximately 75 minutes. We first showed an animation of a robot going through a bookshelf and counting the number of times it comes across a book with the letter E on it. Then we showed them the same animation but with elements of the code superimposed on the animation. Finally, we showed then a third animation code-tracing a program containing a for-loop, that represents the same idea as in the first two animations. All animations were accompanied with a narration explaining the steps taken as well as making connections between each video. Figure 3.4 shows a snippet of the narration script.

Participants were prompted to verbally answer questions about the videos[2]. These ranged from questions about the specific design of the animations, to conceptual questions about their knowledge of the material presented in the animations. All sessions were recorded and the transcripts analyzed; the researcher also took extensive notes.

[2] Verbal prompts were not part of the main study described in Chapters 4 and 5. In the main study, prompts were embedded in the tutor and participants typed responses into textboxes in the tutor interface.

Figure 3.4

Excerpt from Narration Script for the First Animation Video in the Pilot

Script:

Imagine the following scenario:

1. On the left we see a robot (in the form of a robotic arm). At the top, we see a shelf with books on it. Each book contains a label consisting of a single letter written on it. The robot needs to use a magnifying glass to see the label because it is small.

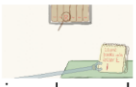
   Let's say we want to give the robot instructions to count the number of books in this shelf that are labelled with the letter E. Before it starts, this 'count' is zero, because so far it has not seen any books labelled E.
   a. To accomplish this task, the robot will need to use the magnifying glass to look at each book one by one in order.
   b. For each book it looks at, it will go to its worktable, shown below the shelf, and follow the instructions on the clipboard.
   c. The instructions tell the robot to count each time it comes across a book with the letter E. This is important: the robot cannot remember instructions like humans, so it forgets them right after it follows them – so it has to come back to this work table each time it sees a new book on the shelf.
2. As we said, the robot can only see the letters on the books one at a time and uses a magnifying glass to see each label.
   a. The first book has the letter W on it.

(Main narration ends at this point. Text in *italic* below is a description of what is happening in the animation and it does not refer to narration).

10. *Robot goes to shelf and moves magnifying glass to the next book.*

    What letter does the current book have on it? (answer: R)

11. *Robot arm moves to the instructions on the clipboard.*



    What will the robot do now? (answer: look at magnifying glass and see what it says. Compare the letter on the book to the letter in the instructions on the clipboard).

12. *Robot does the comparison – looks at magnifying glass, then looks back to clipboard.*

    Will the robot update the count on the clipboard? If yes, what will it write?
    (answer: since the current book has the letter R on it, the robot does not change the value of count and it stays at 1).

13. *Robot arm goes back to shelf. Moves magnifying glass over to the next book. Letter on book is E. Robot arm goes to instructions on the clipboard. Compares instructions with current position of magnifying glass.*

    Will the robot update the count on the clipboard? If yes, what will it write?
    (answer: since the current book has the letter E on it, the robot adds 1 to count. The count is now 2.)

Note. Text in purple denotes prompts to be answered by participants.

The feedback indicated that participants were able to understand the main components of a loop (i.e., that elements of the list are checked one at a time, count only updates if the condition is met, and the computer must return to the instructions each time it goes through the loop). However, participants also highlighted some parts that were unclear. The main one was that they were not sure about the role of the magnifying glass. The magnifying glass was intended to symbolize the idea that the robot is currently only looking at one letter. In addition, in the pilot animations, once the robot had seen the letter on the current book with the magnifying glass, it would move to the instructions on the notebook, and would then look back at which letter the magnifying glass was showing. Participants had a hard time understanding this notion - that the robot does not remember what the current letter is and needs to go check what current letter the magnifying glass is looking at again.

To implement this feedback and improve the design of the animations, we revised the narration scripts to include an explanation of the role of the magnifying glass. We also streamlined the animation itself, reducing excessive frames containing back and forth movement between the notebook and the magnifying glass. Instead of showing it explicitly, we explained in the script that the robot needs to check what the magnifying glass is currently showing (but did not animate it to reduce unnecessary complexity). This simplified the animation and made it more straightforward to understand, in combination with the narration.

3.7 Fading Between Stages

As mentioned previously, the concreteness fading method relies on gradual fading between three stages: concrete (Stage 1), semiconcrete (Stage 2), abstract (Stage 3). After

having refined - by brainstorming, designing, and piloting - the most appropriate concrete representation for Stage 1, we next implemented the design for fading from Stage 1 to Stage 3, as per the guidelines of the concreteness fading approach. Fyfe et al. (2014) proposes that fading is achieved by *"starting with a concrete, recognizable format, then gradually and explicitly removing context-specific elements, to generate a more abstract representation"*. This fading process allows the student to link their knowledge from the concrete to the abstract material.

Unfortunately, there is no clear consensus on how this fading should occur (e.g., what does 'gradually' mean; how much information should be removed from each stage). The implementation of fading differs between researchers. For instance, Fyfe et al. (2015) go from physical monkey/frog puppets in stage 1, to drawings of a monkey/frog in black and white in stage 2. Hence, the representation changes in two ways – first colour is removed, and it becomes a graphic representation (2D as opposed to 3D). Trory et al. (2018) use a physical environment for the concrete stage and for stage 2 use a graphic representation, but with a 3D perspective and the same colours as used for stage 1, with added labels. Given this variation, it is unclear what exactly is the 'correct' approach, that is truest to this method, since both examples are considered concreteness fading. To get a better understanding of the various fading options, we outlined the various ways to fade from the concrete to the abstract representation, in three stages, shown in Figure 3.5 (note that these are not all the possible combinations, but the curated main ones). The images for stage 1 and 2 in Figure 3.5, are screenshots of what are animations in the actual lesson in the tutor.

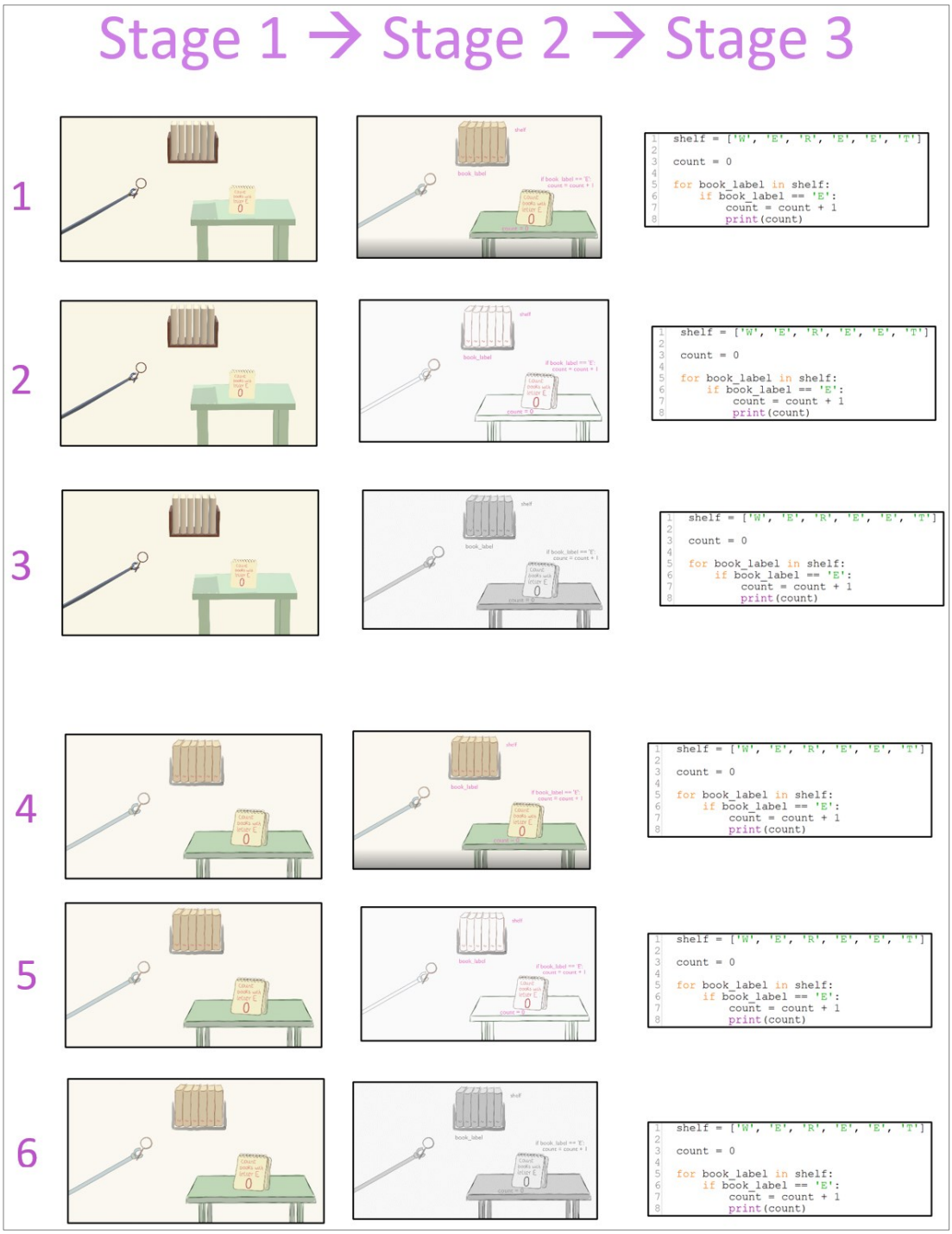
For options 1-3, stage 1 (concrete) is a 3D model of a virtual environment. The goal is for the scene to resemble a real-world scenario as much as possible. Thus, each object is

made of its corresponding material – the robot arm is metal, the magnifying glass and bookshelf are wooden, the table is glossy, the notebook is paper, etc. In option 2, in the second stage, colour has been removed (as opposed to stage 2 in option 1), whereas stage 2 in option 3 is completely black and white. It is the same idea for options 4-6, but this time, the concrete stage 1 in each of them is no longer a virtual 3D environment. Rather, it is a drawing of a 3D environment (a 2D representation of a 3D environment).

The layout we chose was option 5, for the following reasons. After reflection, stage 1 in options 1-3 with the virtual 3D environment is not that different from stage 1 in options 4-6. It (stage 1 in options 1-3) brings across the same information as the 2D drawing in stage 1 of option 4-6 (the 2D drawing has 3D perspective). It would also take much more effort to create the 3D environment. This eliminated options 1-3, and we were left with 4-6. For option 6, we felt that the black and white stage 2 did not do justice to the fading, did not add anything new, and might even be hindering and confusing because of the darker areas. As for option 4, stage 2 seemed too similar to stage 1, and the fading was a rather small step (the only difference between both stages is that labels are added). Hence, this left option 5, which has a good balance of fading between each stage. In other words, the fading was gradual, without making too big of a jump between each stage.

Figure 3.5

Possible Combinations of Fading Between Concrete and Abstract Stages



Note. Option 5 was the selected option used in the present study.

As far as fading in option 5, stage 2 has the same elements as the concrete stage 1, but colour – a perceptual detail – is removed and code elements are added (important). The final stage 3 just has the code elements (this is the abstract representation). The middle stage 2, is the link between stage 1 and 3 and is hence indispensable. This is a crucial part of concreteness fading.

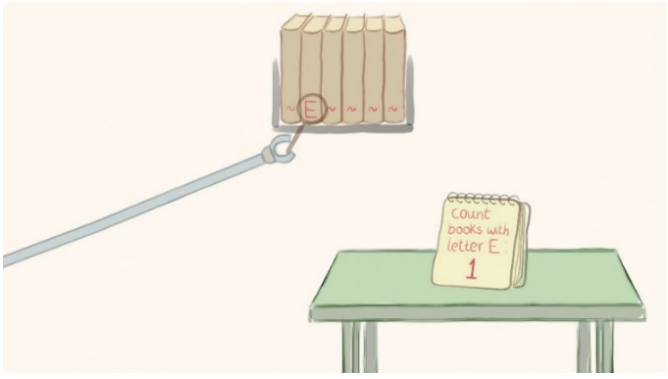
3.7.1 Detailed Description of Each Stage

As mentioned, the final chosen layout was option 5 (see Figure 3.5). We now describe it.

Stage 1: In stage 1, the animation is coloured and shows a simple scene consisting of a robot arm holding a magnifying glass, a bookshelf with books on it, and a table with a notebook that has the instructions “count books with letter E” on it (Figure 3.6). There is a narration explaining the scenario and goal – for the robot to go through all the books on the bookshelf and count how many books it comes across that contain the letter E on them (see Stage 1 script in Appendix A.1).

Figure 3.6

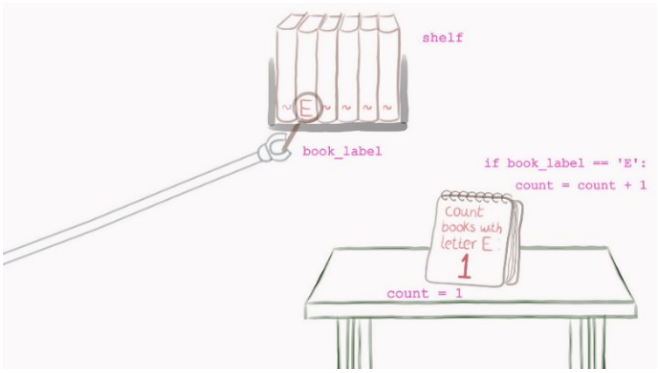
Stage 1 (Concrete)



Stage 2: Stage 2 shows the same animation as in stage 1, but perceptual details such as colour are removed (Figure 3.7). There is also an addition of Python code elements. This is the middle stage that makes a bridge between the concrete and the abstract (code). The narration explains the links between the code and the elements of the animation (see Stage 2 script in Appendix A.2).

Figure 3.7

Stage 2 (Semiconcrete)



Stage 3: Stage 3 shows an animation of a code trace of a Python program (Figure 3.8). This program portrays the situation with the robot and bookshelf seen in stages 1 and 2, but now only Python code is left visible. The narration points to the similarity between the features of this code and the previous two animations. The code elements that were present in stage 2, are the exact same elements as in the program in stage 3.

To make the links between stage 1 and stage 3 explicit, stage 3 contains snippets of the stage 1 animation. The narration explains the relationship between an element of the code and its related analogy in the stage 1 animation with the robot (see Stage 3 script in Appendix A.3). For example, in Figure 3.8, we can see a moment in the animation where it explains that the bookshelf in the stage 1 animation (lower right corner) represented the highlighted ‘shelf’ variable in the code.

While stage 3 only contains an animation of the abstract code, the animation does reference the previous animation with the robot. Thus, technically this third animation (stage 3) in the concreteness fading condition does not contain only code (since it makes reference to the animation in stage 1 and includes non-abstract elements). Nevertheless, making links between stages is an essential part of implementing the concreteness fading approach (as discussed in Section 2.2). There could be a fourth stage after stage 3 that would contain only the code, and no reference to the previous animations. However, this is not standard in concreteness fading studies and thus we did not implement this fourth stage.

Figure 3.8

Stage 3 (Abstract)

```
1  shelf = ['W', 'E', 'R', 'E', 'E', 'T']
2
3  count = 0
4
5  for book_label in shelf:
6      if book_label == 'E':
7          count = count + 1
8          print(count)
9
10 print("We are done!")
```

3.8 Abstract-Only Lesson

Section 3.7.1 offered a description of the three consecutive stages (concrete, semiconcrete, and abstract) presented as animations in the concreteness fading lesson. We also created an abstract-only animation that included only the abstract stage (see Table 3.2). The video presented in this stage was almost identical to the video in the abstract stage (stage 3) of the concreteness fading condition (code with narration and program

explanation). The only difference was that in the abstract-only condition, no reference was made to the previous two animations (concrete and semiconcrete) because the participants have not seen those videos. The abstract-only condition was the control condition because the programming concept is introduced in a similar manner to standard instruction – by code-tracing a program and explaining its various components and flow of execution.

Table 3.2

Study Design

	Stage 1 →	**Stage 2 →**	**Stage 3**
Concreteness Fading	Concrete (animation)	Semiconcrete (animation with code)	Abstract (only code)
Abstract-Only	N/A	N/A	Abstract (only code)

Note. Table shows the content of each condition: Concreteness Fading and Abstract-Only

Chapter 4: The Concreteness Fading (CF) Tutor

Two versions of a computer tutor were created, one that used the concreteness fading instruction and a control version that used the standard approach for presenting the for-loop instruction (i.e., abstract-only instruction without concreteness fading). To build the tutors we used the Cognitive Tutor Authoring Tools (CTAT) framework[3] (Aleven et al., 2016). We will refer to the tutors as the *CF Tutor* and *Standard Tutor* when we need to differentiate them, and simply as the *Tutor* otherwise (the control version of the tutor (*Standard*) was identical except it did not contain the fading paradigm – explained below). The tutor is comprised of 14 or 20 (depending on the condition) windows showing instructional materials. The windows are displayed one at a time and students move through them in consecutive order by clicking a 'Done' button. Each window is different – some contain only text, others contain a video, and others contain text boxes requiring student input, with the option to check the answer. We first describe the Tutor in detail, then explain the tutor-creation process.

4.1 Tutor Components

The tutor consists of 4 main elements: a tutorial, animation videos, prompts, and activities.

4.1.1 Tutorial

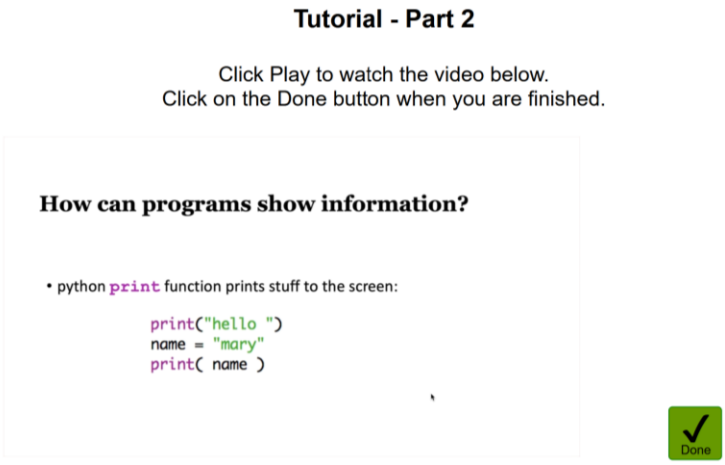
The Tutor first presents a short tutorial on programming, in the form of two videos shown in separate windows, with a break in the middle (part 1 of the video is 8 minutes, followed by a 3-minute break, followed by part 2 which is 13 minutes) (Figure 4.1). The tutorial provides an introduction to programming, including variables, strings, numbers,

[3] http://ctat.pact.cs.cmu.edu/

lists, the print function, conditional statements (if-statements and proper indentation), and code traces several examples of programs. These concepts are needed to understand for-loops, which is the concept that the lesson in this study aims to teach. Therefore, for-loops are not included in the tutorial.

Figure 4.1

Tutorial Window in the Tutor

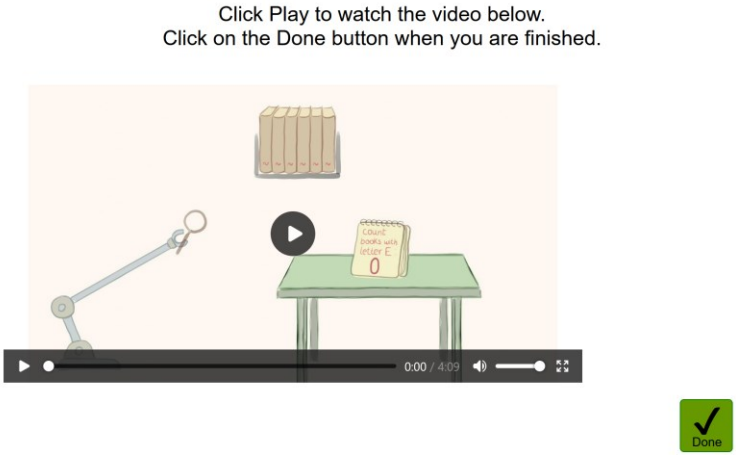


4.1.2 For-Loop Lesson

The next series of windows in the Tutor corresponded to the for-loop lesson. The lesson had two parts: animation video(s) and prompts about the video, respectively, shown in separate screens. The lesson differed between the two tutor versions. Specifically, the CF Tutor included three animation videos shown on three separate windows and the Standard Tutor included one animation video shown on one window (see Chapter 3 for a description of each). Figure 4.2 shows a screenshot of the first lesson video in the CF Tutor.

Figure 4.2

First Animation Video in the Lesson in the CF Tutor



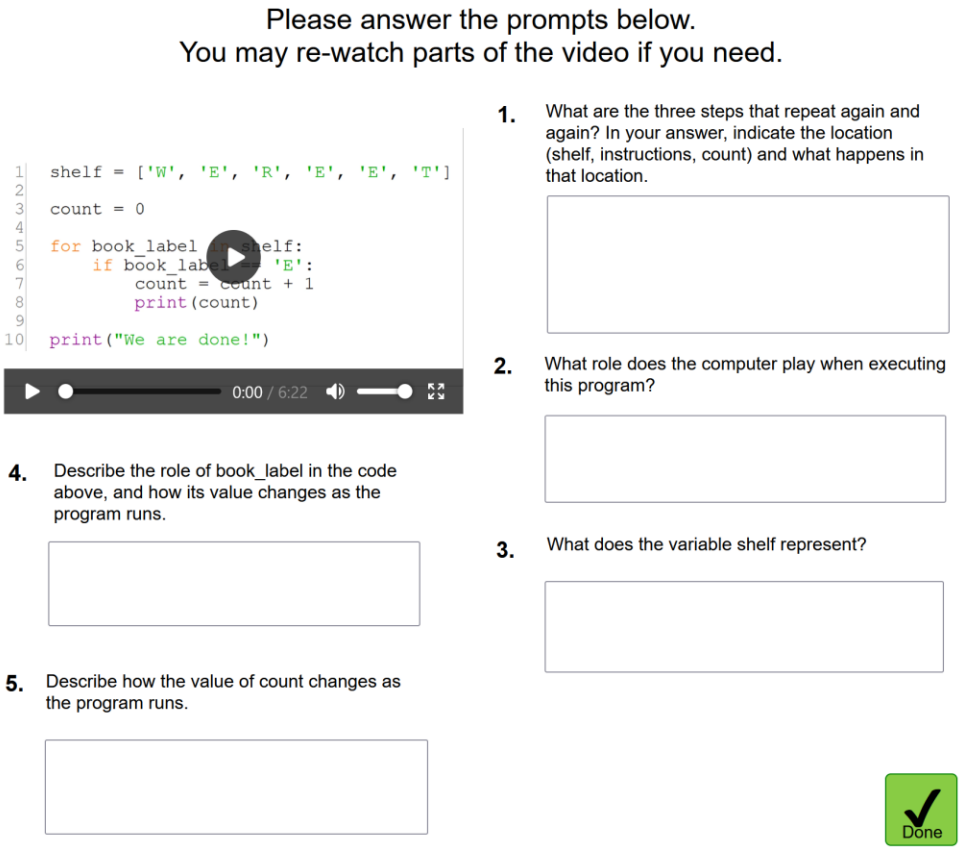
Each video was followed by a screen with questions about the lesson that prompted students to reflect on the content (see Figure 4.3). Efforts were made to equalize the number of prompts between the two tutor versions, but the CF Tutor had more prompts given that it had two extra lesson videos (*CF Tutor # prompts* = 14; *Standard Tutor # prompts* = 10) (see Appendix B for all prompts present in both Tutors). The Standard Tutor included a subset of the same prompts as the CF Tutor. The prompts were designed to encourage students to actively and constructively think about the concepts, instead of passively receiving information solely from watching the videos. Students' answers to these questions were free form and were not graded by the Tutor. These prompts were embedded in the tutor and students typed their responses (rather than providing verbal responses as in the pilot study described in Section 3.6). When answering the questions, the Tutor provided access to the lesson video just watched (Figure 4.3) but not all answers were explicitly provided by the video(s). Initially, we wanted to include a button in the interface that would bring the student to the previous screen to review the video, but we decided that it would

be better to have the video on the same screen as the prompts so that students do not have to split their attention and flip back and forth between screens.

The Tutor required that questions be answered before moving on (accomplished by pressing the Done button). If a user tried to move on before submitting a response, a popup alert message appeared informing them that they had to submit an answer to each question.

Figure 4.3

Standard Tutor Interface Showing Prompts to Answer



Note. When answering prompts, access to the animation video is provided

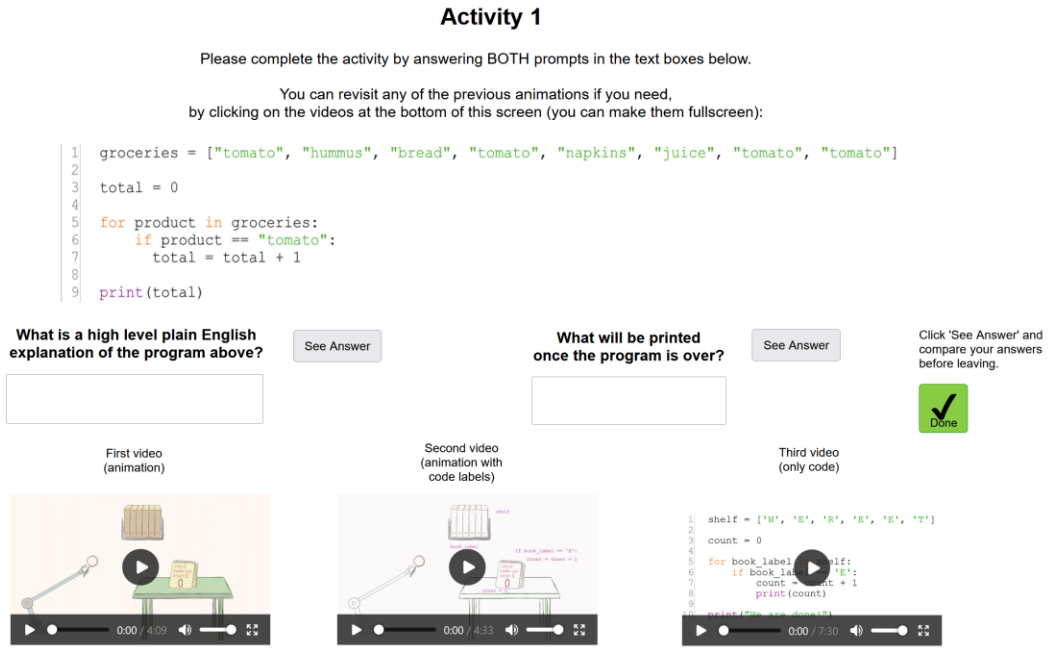
4.1.3 Tutor Activities

Following the lesson, the Tutor provided two code-tracing activities that required users to predict the output of a program containing a for loop. The activities were designed to offer practice and knowledge consolidation opportunities (both versions of the tutor used the same activities). For both activities, the Tutor provided access to the lesson videos previously watched. After an answer is submitted, users could ask the Tutor to see the correct answer by clicking the 'See Answer' button. If they clicked the 'See Answer' button before submitting an answer, an alert message pops up telling them to first complete the activity before checking their answer.

Activity 1 was a near-transfer exercise, similar to the lesson. Specifically, the exercise included a program with a for-loop and the program was isomorphic to the one in the lesson but with different variable names. Participants were asked to give a high-level explanation of the program and predict its output (Figure 4.4).

Figure 4.4

CF Tutor Activity 1

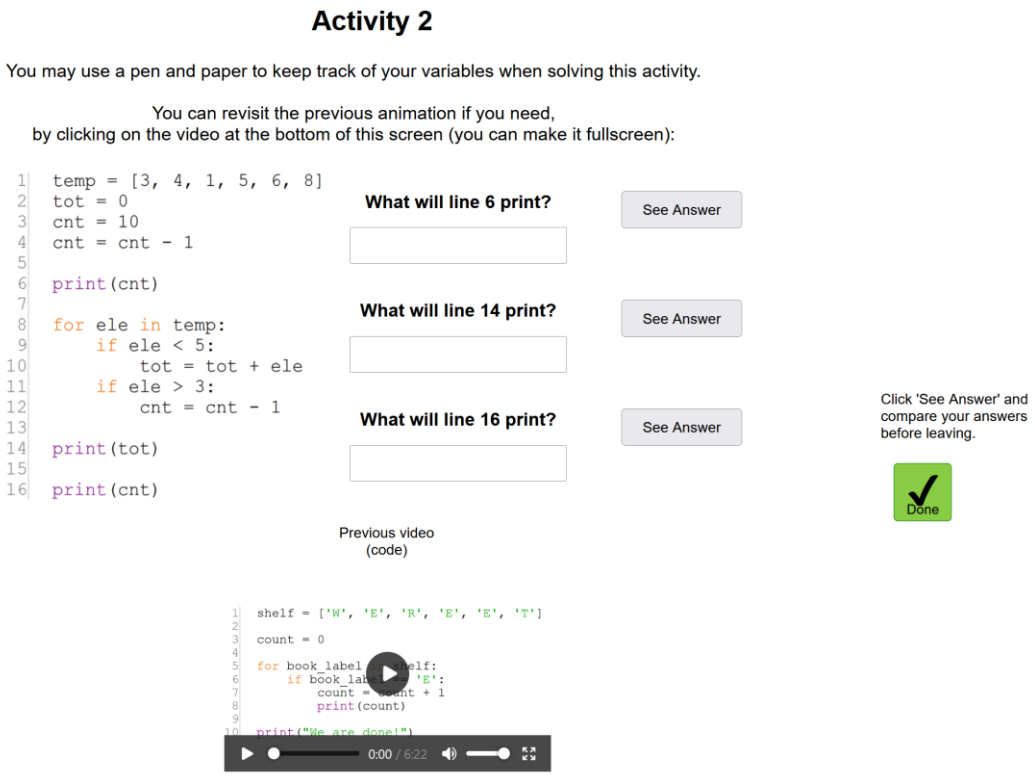


Note. Activity 1 – near-transfer problem with ‘See Answer’ buttons to compare answers

Activity 2 was a far-transfer exercise. While it also contained a program with a for-loop, it used numbers instead of strings and the variable names were less clear, making it harder to solve the problem with educated guesses (Figure 4.5). This activity is more difficult because it requires a strong understanding of the flow of execution of the program (how the for-loop works and the importance of the indented blocks); moreover, there are two different conditions checked in the loop (if-statements) and each condition has its own output.

Figure 4.5

Standard Tutor Activity 2



Note. Activity 2 – more difficult problem requiring a stronger understanding of the flow of execution of the program. Also note the difference between the Standard and CF Tutor interface for the Activities – the CF Tutor gives access to all three previous animations, compared to the Standard Tutor which only has one animation video.

Both activities were code-tracing exercises. We considered other options for the activities, such as having the participants modify existing code, fill in the blanks, or put lines of code in order (Parsons puzzle problem). However, modifying code or filling in the blanks require writing actual code, which is a different skill (Xie et al., 2019) and not the focus of the present lesson, which focuses on program and concept comprehension. These skills are better tested with code tracing exercises (Xie et al., 2019).

4.2 Building the Tutor

Building each version of the tutor involved creating the interface, linking each window of the tutor interface to its own behavior graph, and deploying the final tutor online.

4.2.1 Creating the Tutor Interface with the CTAT HTML Editor

The first step involved creating the interface for each of the Tutor's windows with the 'CTAT HTML Editor'. This editor allows a human "author" to drag and drop components onto an interface-creation area, alleviating the need to program the HTML widgets. Once the file is saved, the Editor automatically creates the corresponding CSS and HTML files. The HTML file contains the raw HTML code specifying the look and behavior of the elements in the interface for a given window.

In some cases, we needed to manually add Javascript functions to the HTML file to implement customized features. For instance, clicking the 'See Answer' button and having the answer appear was not built into the CTAT framework and required an author-defined function. The HTML files also required editing to format the look and feel of the interface in a given window (e.g., to make certain widgets initially invisible).

4.2.2 Specifying the Tutor Reactions with the CTAT Behavior Graph Editor

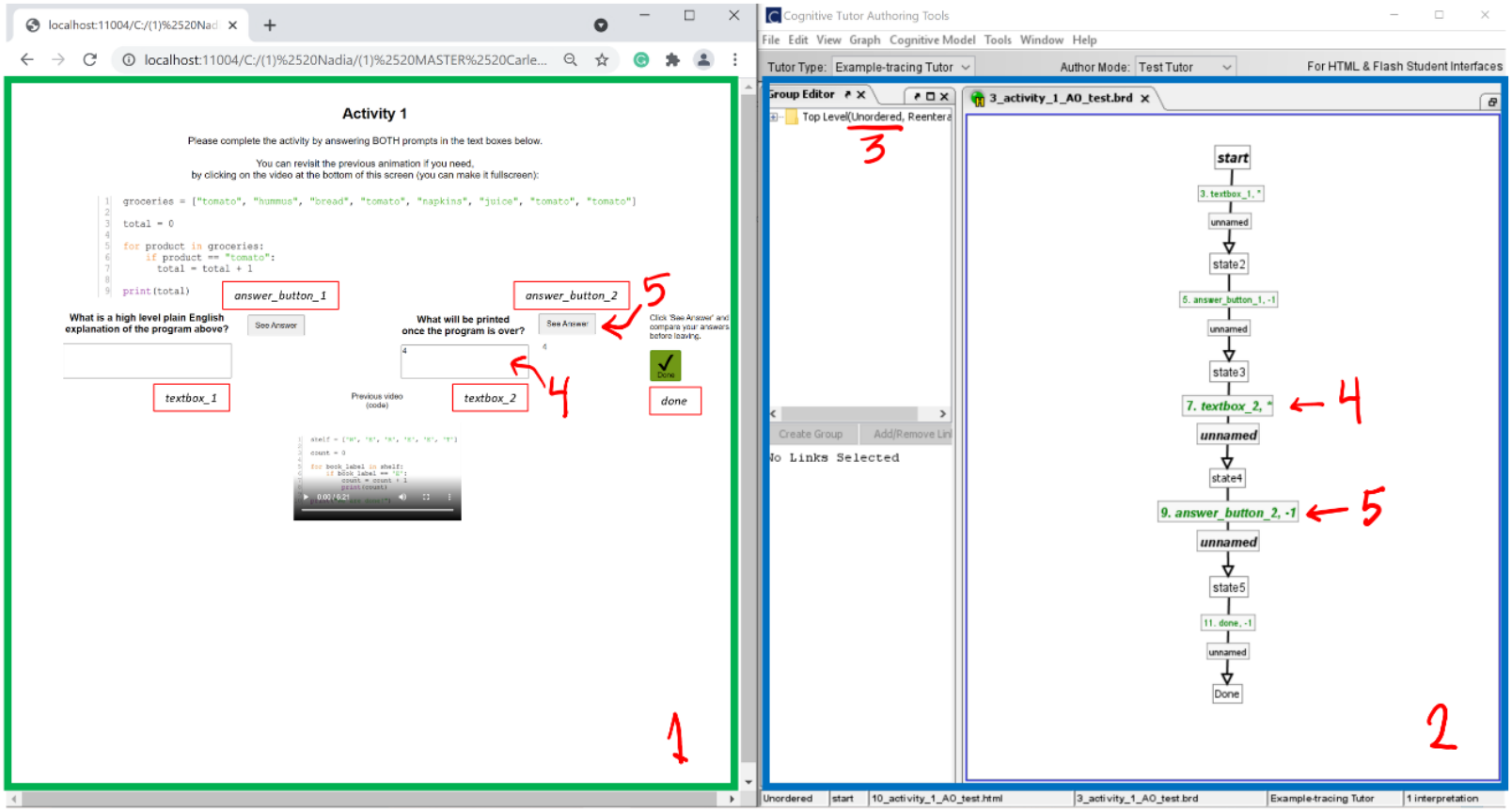
Once all of the HTML files were created, corresponding to each window in the Tutor's interface, the next step involved building a *behavior graph* using the CTAT behavior-recorder tool. A behavior graph allows the author to specify tutor behaviors in response to student actions (Aleven et al., 2016). To build the graph, the author has to demonstrate all the steps that a student would need to perform in a given tutor window; The CTAT behavior recorder records the demonstrated behavior as a behavior graph

(Aleven et al., 2016). Each HTML file (corresponding to the interface of a window in the Tutor) is paired with its own behavior graph that encodes actions a student needs to perform in a given window.

Figure 4.6 shows an example of an interface window in the Tutor (left) and its corresponding behavior graph (right). They are paired together. The behavior graph includes the various interactive components of the tutor interface for this window, including *textbox_1*, *textbox_2*, *answer_button_1*, *answer_button_2*, and *done* (these are shown in green in the blue box on the right, Figure 4.6). These labels refer to their corresponding components in the interface (see labels in red boxes in Figure 4.6 left, added to the interface for demonstration purposes). The *textbox* elements capture the entry of a solution step, while the *answer_button* elements capture when the user clicks the button to check their answer. In the upper left corner of the behavior graph (see red label *3* in Figure 4.6), we specified that the behavior is *Unordered*, which means that participants can complete the steps in any order. The only exception to this is that the Done button must be pressed last, because it indicates that all required steps have been completed.

Figure 4.6

HTML Interface (Left) and its Corresponding Behavior Graph (Right)



Note. Interface and Behavior graphs of the Tutor built using CTAT

CTAT allows the author to test the tutor window associated with the behavior graph locally on their computer, accomplished by performing actions in the tutor window in so-called 'test mode'. In test mode, when the author clicks on a widget or provides a solution for an interface element that is in the behavior graph, the graph entry becomes bolded. For instance, suppose the author first inputs their answer to *textbox_2*, and then clicks on *answer_button_2* (before completing *textbox_1*). These behaviors are bolded in the behavior graph (see 4 and 5, right, Figure 4.6). The other elements have not yet been completed and they are therefore not bolded yet in the behavior graph.

We built two tutors, one for the experimental condition (CF Tutor) and one for the control (Standard Tutor). As noted above, the only difference between the two tutors corresponded to the lesson, namely that the CF Tutor animations included a concrete and intermediate phase (with prompts about these animations) while the Standard Tutor did not. Both tutors had an animation about the abstract stage describing the mechanics of a for loop and corresponding prompts.

4.2.3 Deploying the Tutor

A CTAT tutor is essentially a sequence of HTML files corresponding to the tutor's windows, each paired with a behavior graph that determines the tutor's responses to student actions for that window. The final step involves deploying the tutor, which performs two important functions. First, it integrates all the separate windows so that students can move from window to window by clicking the 'Done' button. This button is what tells the software to move to the next window. The second important function deployment accomplishes is that it makes the tutor available online, meaning that students can easily access it without having to download any software (beyond a web browser).

We deployed the CF and Standard Tutor on the platform TutorShop[4]. This platform allowed us to store both versions of the tutor (experimental and control conditions), assign participants to their condition, and log all interactions within the tutor. Once each tutor was deployed on TutorShop, it had to be tested in that new online context (recall that each window was tested locally on the researcher's computer). Some elements no longer worked correctly in the online context (e.g., video files resized incorrectly, certain buttons did not work as intended); bugs were corrected by making changes to each relevant HTML file accordingly, which was then reuploaded to be tested again. Once all issues were fixed, the final version was linked to the DataShop platform[5], which logs student interactions with the tutor.

DataShop is an open repository of educational technology data (Koedinger et al., 2010), which records each interaction with the interface such as clicking buttons, pressing play/pause on the videos and recording the exact time the button was pressed, the answers to the prompts inputted in the textboxes, the duration of each interaction, as well as the participant ID, experimental condition, and date and time of the experiment. These elements allow for the analysis of additional features in the data (e.g., do participants who spend longer on each prompt, perform better on that same prompt?).

[4] https://school.TutorShop.web.cmu.edu/
[5] https://pslcDataShop.web.cmu.edu/index.jsp

Chapter 5: Study Methodology

We evaluated the tutor using an experimental study. In this chapter, we present the study methodology.

5.1 Participants

The participants were 52 university students (37 female, 14 male, 1 undisclosed). Participants were aged between 18-52 (*average age* = 23).

Participants were recruited through social media ads (Facebook groups and societies, Instagram stories, Discord posts), word of mouth, and through the SONA online recruitment system. Participants that signed up through SONA were CGSC 1001 students, and they received a 2% credit for their course after completing the study. If participants did not sign up through SONA, they received a $25 compensation instead. Participants either did not have any previous programming experience, or had taken at most one university-level programming course. Participants who had taken more than one university programming class were not eligible because they had too much programming knowledge.

5.2 Materials: Demographics, Pretest, and Posttest

A brief questionnaire collected demographics information about participants (see Appendix C).

A pretest and a posttest were used to measure domain knowledge. The pre- and posttest were identical except that some variables in the posttest questions were changed. Both tests included 10 code tracing exercises, 1 code explanation exercise, and 1 code writing exercise (see Appendix D). For each code tracing exercise, participants had to answer what the final output of the given program would be (based on the print() statements

in the code). All questions had only one correct answer, except for the final code writing problem that could be solved in several ways.

The first five questions were on the fundamentals of programming, related to what was taught in the tutorial. This includes updating variables (string and integer types), conditional statements, and understanding the flow of execution of the program (paying attention to indentation). The next 7 questions were related to the lesson in the tutor, meaning they involved for-loops. Three of these questions were near-transfer problems, very similar to the ones in the lesson and activities. The remaining 4 questions were more difficult as they involved multiple conditional statements inside a for-loop, more complicated execution flows that needed attentive tracking, or a programming exercise to test code-writing knowledge.

The posttest in the concreteness fading (experimental) condition also contained one extra question on the design of the animation videos: *"Do you think that seeing the first two animations (with the bookshelf and robot) was helpful to understand the final third animation (with the code)? Could you please elaborate on what you liked/did not like about the animations. Be specific."* This question was designed to collect qualitative information about the participants' perceptions of the lesson beyond their pre- and posttest scores. The posttest in the abstract-only (control) condition did not contain this question since participants did not watch the same animations in this condition (see Section 3.8).

The study materials also included the tutorial and lesson embedded in the CTAT tutor, described in Chapters 3 and 4.

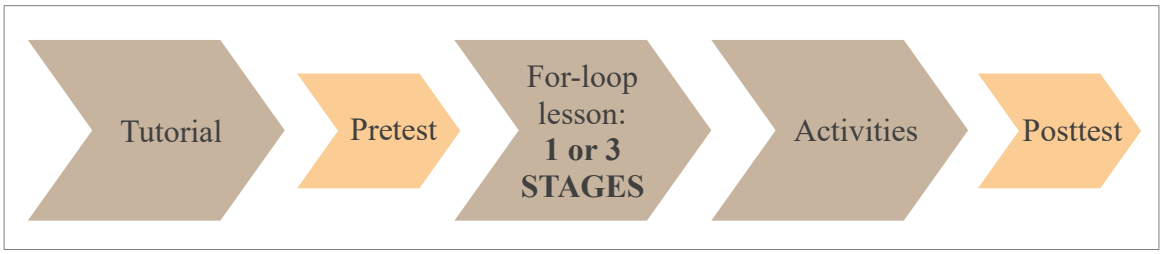
5.3 Study Design and Conditions

The experiment used a between subject design with two conditions – concreteness fading (experimental) and abstract-only (control). We did not include a concrete-only condition as prior work has consistently shown the concrete-only condition as not effective, compared to abstract-only and concreteness fading conditions (Fyfe et al. 2015; Trory et al. 2018; Goldstone & Son 2005; Kaminski et al. 2008a, 2008b; McNeil & Fyfe 2012). Each participant was assigned to a condition in a round robin fashion. There were 26 participants in each condition and the number of participants that had already taken one university-level programming course was similar across both conditions (6 in experimental condition, 7 in control condition).

The concreteness fading group interacted with the CF Tutor, while the abstract-only group used the Standard Tutor. Both tutors contained the tutorial at the beginning, the lesson in the middle, and two activities at the end, all embedded in the respective online computer Tutor. The conditions differed in the lesson itself, as described in Section 3.8. Even though careful attention was paid to equalizing time on task as well as equalizing the number of prompts after each animation, the control condition was shorter than the experimental condition, because it only contained one animation video (as opposed to three) in the lesson. On average, it took participants in the experimental condition 17 minutes to go through the extra two videos and their accompanying questions (the average total time to complete the study being 58 minutes in the control condition and 68 minutes in the experimental condition, excluding the pre- and posttest (so this is the time taken to complete the tutorial and the lesson in the tutor that contains the animation videos, prompts, and activities)).

Figure 5.1

Diagram of the Study Procedure



Note. The for-loop lesson contains animation videos and prompts. The Concreteness Fading condition has 3 stages of animations and the Abstract-Only condition has 1 stage. All sections (brown arrows in the diagram above) except for the pretest and posttest are embedded in the Tutor.

5.4 Procedure

Figure 5.1 shows a diagram of the main procedure steps. The study was approved by the Carleton Research Ethics Board. It was conducted online through Zoom by the thesis author (the online format was dictated by the COVID-19 pandemic preventing in-person meetings). When a participant signed up for the study, they received an email containing additional information about the study, as well as the Zoom link and password to join at their designated time and date. Email reminders were sent to participants the day before and the day-of the experiment. Prior to every session, the participant was assigned a participant ID that was logged in the researcher's logbook, along with any additional notes during the experiment; duration of the whole study and time on pre- and posttest were also logged. The study took no longer than 2 hours. Participants entered their ID at the start of the tutor lesson, as well as on both the pre- and posttest forms.

Participants first read and signed the consent form (see Appendix E). Next, participants were asked to confirm whether they had taken a university-level programming course before, and if yes to state which one. Participants who had already taken a course

previously were notified that some of the material might seem familiar. The study procedure was then outlined, including the length of the tutorial videos. The researcher emphasized that no programming background was required for this study.

Next, the main phase of the study began. The audio and screen were recorded, but participants' cameras were turned off to preserve privacy. The researcher's camera was also turned off when participants were completing the main study components to allow them to work at their own pace. However, the researcher was present the whole time to address logistical questions about the interface or clarifications (questions about the domain itself were not answered). Participants were asked to share their screen (and close any tabs they did not want to be visible in the recording before sharing), go to the website containing the Tutor, and login with the username and password, all provided in the Zoom chat. Participants were told that they could take notes if they want to because the tutorial material would not be available afterwards.

After viewing the two tutorial videos (with a 3-minute break in between), participants stopped sharing their screen and were given a link to the pretest. They had a maximum of 15 minutes to complete the pretest. They were told that to search the internet for answers at any point during the study was not allowed (even though the answers could not explicitly be found online). Participants were encouraged to do their best, were reminded that this was not an exam, and that if they did not know the answer they could put 'IDK' (I don't know).

When participants were finished with the pretest, they shared their screen again, and completed the main portion of the tutor lesson in their browser.

The final step in the study involved the posttest. Participants were once again told to stop screensharing, and go to the posttest link provided to them in the chat; participants were informed that the posttest would look similar to the one they already filled out at the start, but to do their best and not blindly guess. They had 20 minutes to complete the test. After this, participants were debriefed, their questions were answered, and information regarding the payment was provided (if applicable).

Chapter 6: Study Results

The following research questions guided the analysis:

1. Does teaching a target programming concept using the concreteness fading approach result in higher learning gains and transfer than regular instruction?
2. What are participants' opinions on using concreteness fading to teach a target programming concept?

6.1 Learning (Concreteness Fading vs. Abstract-Only Instruction)

Learning was operationalized as the difference between post- and pretest scores (posttest – pretest). As described in Section 5.2, the pre- and posttest contained different types of questions, including fundamental programming questions, near- and far-transfer questions, as well as a code-writing question, as follows:

- Question 1-5: fundamentals covered in tutorial, do not include for-loops
- Question 6-8: near-transfer, similar to the lesson, do include for-loops
- Question 9-11: far-transfer, more difficult conceptually
- Question 12: code-writing

All pretests and posttests were marked according to a rubric. Questions 1-5, and 7 were scored as either correct or not (1 point for correct; 0 for incorrect). Questions 6 and 8-11 were scored on a 2-point scale, with partial points possible. The final Question 12 was scored out of 5 because it was a code-writing exercise that allowed for variety in its solution. Participants were not informed about this grading system.

Two participants with pretest scores higher than 95% were removed from analysis, as they were essentially at ceiling (one from each condition). This left 25 participants per condition ($N = 50$). Table 6.1 presents the pretest and posttest scores as well as the learning gains (also referred to as 'difference' scores below) for all question types.

Table 6.1

Mean Pretest, Posttest, and Difference Scores

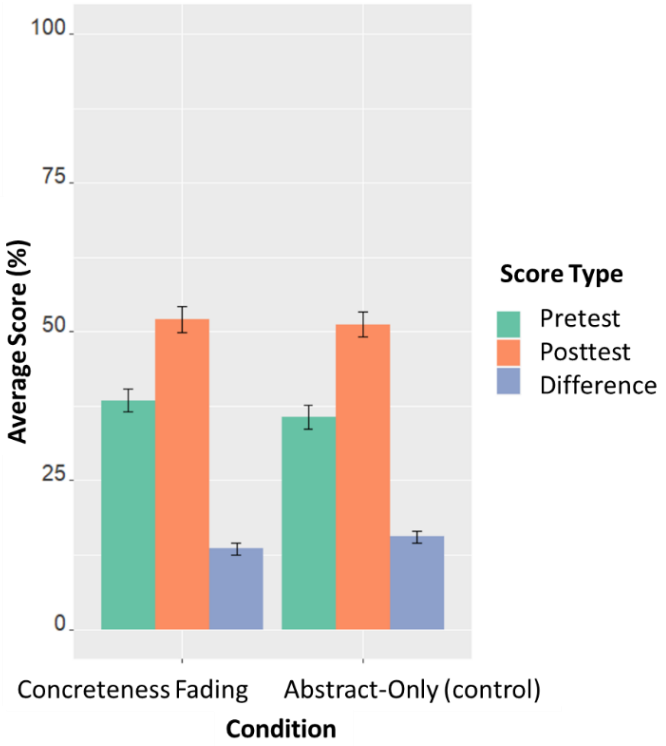
		Concreteness Fading *n = 25*		**Abstract-Only (control)** *n = 25*	
		Mean	**SD**	**Mean**	**SD**
All questions (/21)	Pretest	8.08 (38.48)	4.82	7.50 (35.71)	5.02
	Posttest	10.92 (52.00)	5.59	10.76 (51.24)	5.32
	Difference	2.84 (13.52)	2.56	3.26 (15.52)	2.57
Fundamental (/5)	Pretest	3.04 (60.80)	1.14	2.32 (46.40)	1.09
	Posttest	3.24 (64.80)	1.16	3.26 (65.20)	1.16
	Difference	0.20 (4.00)	0.65	0.94 (18.80)	0.83
Near Transfer (/5)	Pretest	1.94 (38.80)	1.91	1.48 (29.60)	1.76
	Posttest	3.30 (66.00)	1.81	3.36 (67.20)	1.86
	Difference	1.36 (27.20)	1.64	1.88 (37.60)	1.83
Far transfer (/6)	Pretest	1.54 (25.67)	1.38	1.46 (24.33)	1.30
	Posttest	2.10 (35.00)	2.08	2.26 (37.67)	1.74
	Difference	0.56 (9.33)	1.78	0.80 (13.33)	1.49
Code writing (/5)	Pretest	1.56 (31.20)	1.89	1.32 (26.40)	1.73
	Posttest	2.28 (45.60)	1.71	1.88 (37.60)	2.00
	Difference	0.72 (14.40)	1.60	0.56 (11.20)	0.93

Note. Mean Pretest, Posttest, and Difference (Posttest – Pretest) scores for both the Concreteness Fading and Abstract-Only (control) conditions, including standard deviation. Raw scores are presented with percentage scores in brackets for all question types.

Overall, average pretest scores were similar between the two conditions and relatively low (38.48% in concreteness fading condition and 35.71% in abstract-only condition). Descriptively, students did learn in both conditions, with average posttest scores rising to 52.00% and 51.24% in the concreteness fading condition and the abstract-only condition, respectively. Figure 6.1 shows the average pretest, posttest, and difference scores for each condition.

Figure 6.1

Average Pretest, Posttest, and Difference Scores



Note. Average Pretest, Posttest, and Difference Scores (Posttest – Pretest) scores in percentage, based on condition. Error bars represent a 95% confidence interval.

We first verified there was no significant difference between conditions in pretest scores. This was the case as reported by an analysis of variance (ANOVA) with condition as the independent variable and pretest scores as the dependent variable, $F(1, 48) = 0.174$, $p = 0.679$, $\eta^2 = 0.004$.

In general, without taking the condition into account, students did learn, as reported by a paired samples t-test with the data aggregated by condition, $t(49) = 8.471$, $p < 0.001$, $d = 1.198$. To determine whether the instruction type had a significant effect on overall learning, an ANOVA with condition as the independent variable and difference score as the dependent was conducted. The ANOVA reported no significant difference between

concreteness fading and abstract-only instruction, $F(1, 48) = 0.336$, $p = 0.565$, $\eta^2 = 0.007$. While NHST does not allow for interpretation of null results, the effect size is very small, suggesting little practical difference between the conditions.

We also analyzed the pretest to posttest percentage gains for each of the four question categories. Combined across conditions, there were significant gains for each question category as reported by a paired samples t-test (*Fundamentals*: $t(49) = 4.874$, $p < 0.001$, $d = 0.689$; *Near Transfer*: $t(49) = 6.585$, $p < 0.001$, $d = 0.931$; *Far Transfer*: $t(49) = 2.953$, $p = 0.005$, $d = 0.418$; *Code-Writing*: $t(49) = 3.487$, $p = 0.001$, $d = 0.493$).

To check if condition influenced these scores, we ran a repeated measures mixed ANOVA with the difference score (entered as percentage) as the dependent variable, the four question categories as the within-subjects factor and condition as the between-subjects factor. The sphericity assumption using Maulchy's test was violated ($p = 0.01$), and hence adjusted using the Greenhouse-Geizer correction ($\varepsilon = 0.846$). Of interest here is the interaction between condition and question category, as it informs on the influence of condition – this interaction was not significant, $F(2.537, 121.760) = 1.034$, $p = 0.372$, $\eta_p^2 = 0.021$. The main effect of question type was significant, indicating that the gains differed between question categories, $F(2.537, 121.760) = 7.143$, $p < 0.001$, $\eta_p^2 = 0.130$. Post-hoc tests with the Bonferroni adjustment showed that learning gains were significantly higher for the near-transfer questions ($M = 32.4\%$) compared to the fundamental questions ($M = 11.4\%$, $p < 0.001$), far-transfer ($M = 11.2\%$, $p = 0.001$), and code-writing question ($M = 12.8\%$, $p = 0.003$). No other comparisons were significant.

6.2 Effect of Animations on Understanding the Abstract Code

Recall that one of the main differences between the two conditions was that the concreteness fading condition included two more stages of instruction inside the tutor that contained an animation. This animation demonstrated a concrete representation of the abstract code in the lesson, and was not present in the abstract-only condition. To obtain insight about participants' opinions about these extra stages, they were asked in the posttest in the concreteness fading condition about their thoughts regarding the animation; *"Do you think that seeing the first two animations (with the bookshelf and robot) was helpful to understand the final third animation (with the code)? Could you please elaborate on what you liked/did not like about the animations. Be specific."* We now report the results using a qualitative approach.

All participants except one (who had already taken a programming course before) reported the animations were helpful to understand the remainder of the lesson that teaches the programming concept using abstract code. For instance, id1 stated that *"The first two animations really helped with understanding how the program works, and make it much easier to understand the third animation."* while id10 wrote that *"Yes it was helpful, as I am not from coding background; I was able to understand a python program as a layman."* One participant, id7, felt the animation helped build a mental model of what the computer was doing: *"Yes I found that animation was more beneficial because it really simplified the thought process of the computer. I liked that it made the entire coding process simpler and easier to understand. It was not as intimidating as the first video instructions which were more complex [referring to tutorial videos]."* Several other participants referred specifically to the concrete grounding in the first animation as helpful. For instance, id14

wrote that *"Having a "real-life" example or at least something that is close to what we can imagine in real life was helpful for me to understand each step that is involved in programming and conceptualize it on a human level."* Along a similar line, id20 reported that the animation was helpful because *"It allowed me to have context and provided me with a visual example of what code does. It made it much clearer to understand what the code was doing."* Id47 also reported liking the analogy because it made things more concrete, i.e., *"I liked the analogy! It helped make the concept less abstract, slowly worked through how each of the steps in the coding related to the analogy, then also explained in detail the names of each of the different components."* Id44 specifically referred to the magnifying glass, because *"it helps visualize what the for loop is doing."*

Lastly, id31 replied to the question of whether the animations were helpful with *"no, it was painfully redundant since the code could tell me everything I needed to know."* This participant had already taken a programming course before but scored 43% on the pretest and 62% on the posttest.

The animations were purposefully designed in a simple way that emphasizes important links between concepts (this is an important part of the concreteness fading approach as explained in Chapter 2). Several participants noted that the animations were easy to understand (id1: *"I liked how easy and clear the videos were to understand."*; id35: *"I liked that the animations were simple and didn't contain unnecessary details."*). Some also liked that the animations took time to explain the concepts (id49: *"The animations took time to explain the video carefully and clearly."*; id10: *"I liked the relaxed speed of animation."*). In contrast, however, other participants found that the animations included too much repetition. For instance, id4 said they *"did not like that sometimes it was*

repetitive and slow, repeating certain concepts that we seemed to already have learned", echoed by id1 (*"I did find it repetitive though, after I understood the concept, it kept explaining it."*). Id19 said the repetition made it *"difficult to focus the whole time."* While id50 also felt the lesson *"could have been somewhat shorter"* they acknowledged this helped to solidify concepts (*"It did however, really hammer home the importance of the loop."*).

In summary, although overall learning increased in this study, no effect of the concreteness fading condition on learning gains was found. Combined across conditions, there were significant gains for each question category and learning gains were significantly higher for the near-transfer questions compared to the fundamentals, far-transfer, and code-writing questions. In addition, most participants found the animation videos helped understand the final animation with abstract code, but some found them redundant at times. Based on these responses, a balance needs to be found between the level of explanation and its repetition. The implication of these results will be further explored in the discussion in Chapter 7.

Chapter 7: Discussion

The present work investigated concreteness fading in the programming domain with university-level participants. Prior research has found the effectiveness of concreteness fading but mainly in the mathematics domain and with a younger population. While students did learn in this study, there was no significant effect of condition on learning gains. We now discuss these findings and future work.

7.1 General Learning Results

The test scores increased from pretest to posttest regardless of condition and the gains were similar in both conditions. This suggests that the lesson was effective in teaching the target programming concept of for-loops, including in the control condition that only included traditional instruction without concreteness fading elements. It may be that traditional instruction is sufficient and that the concreteness fading approach is not necessary for programming instruction. However, it is premature to conclude this given that learning was modest (control: *gain* = 15.52%; experimental: *gain* = 13.52%). Although these gains are not high, previous studies on concreteness fading show a similar pattern (Fyfe et al., 2015; Trory et al., 2018). In general, the learning results in the present research are expected given the programming topic was complex and brand new to most of the participants. Therefore, more work is required to pinpoint where and how students can better be supported when learning difficult programming concepts. An analysis of the transcripts reveals that many participants expressed that the study was difficult. They found the exercises in the tutor lesson as well as the pretest hard in some cases, and based on experimenter notes, several participants really struggled (as evidenced by their activities during the study and test scores after). In sum, the complexity of the topic is one

explanation for why the learning gains were modest. We discuss several additional possibilities below.

Student mindset may have diminished learning in the present study. If participants had general apprehension towards programming or believed that it is a difficult topic before even starting the lesson, their performance could be affected (see Bandura, 1994; Dweck, 2008). Some participants indeed had the notion that they were not good at programming and might therefore have been discouraged to work on the activities. For instance, id29 expressed (after tutorial and pretest) *"That was so confusing! I'm not really good with numbers."* Similarly, id7 said they were nervous about participating and not knowing anything about programming, and at the end noted *"I'm just embarrassed about how little I knew […] this is not my forte".* Mindset was not measured for this thesis to keep the study time under 2 hours but is something to consider in future work. Nevertheless, this was not the case for all participants: several participants stated after completing the study that they had learned a lot, found the lesson fun, and wanted to know more about programming. For instance, id15 said *"Now I have a bit more appreciation for coding",* and id39 asked for additional resources *"Do you have any websites if I want to learn Python and continue?"*

Another potential contributing factor affecting learning could be participant burnout. The study was relatively long, so it is also possible that some participants lost energy as the study progressed, and by the time they reached the posttest, they did not have the motivation to carefully think of their answers. The average time on pretest and posttest was 13 and 11 minutes respectively. This is relatively quick for time on posttest, given that they had 20 minutes to complete it. One way to address study length is to break the study

into two sessions. This, however, creates a challenge of attrition, with participants not returning for the second session.

The analysis for the individual question categories mirrors the overall learning results. Specifically, as reported in Chapter 6, combined across conditions, there were significant gains for each question category in the pre- and posttests, however condition did not have an influence on these scores. Learning gains were significantly higher for the near-transfer questions compared to the fundamental programming questions, far-transfer, and code-writing question. Hence, participants improved the most on near-transfer questions. This makes sense given that the near-transfer problems were similar to the lesson, suggesting that participants were able to apply their knowledge from the lesson on the test. Far-transfer and programming question were intentionally harder and thus it is not surprising that the gains for these questions were smaller than for near-transfer as this kind of transfer is notoriously challenging in general (Dixon, 2012). As for the fundamentals, the lesson did not focus on this, and this may therefore explain why there were no significant improvements from pretest to posttest.

7.2 Why Did Concreteness Fading Not Show a Significant Improvement in Learning?

We now turn to discussing the lack of significant difference between conditions. There are a number of potential reasons for why we did not find evidence that concreteness fading was more effective than traditional instruction. One reason relates to the design of the materials, specifically the animations used in the concreteness fading condition. In section 6.2 we saw that some participants found the animations a bit slow or repetitive, and if they got distracted by this, it would have affected their performance on the prompts in

the lesson. The first two animations were intentionally similar, because following the guidelines of a proper concreteness fading approach (outlined in Section 2.2), overlap between stages is essential. Yet, some participants stated that perhaps both videos could have been combined into one: id25 wrote *"The third animation was really complete and detailed, so it probably would have been sufficient to have the first and last ones only. But they all were very helpful."* Id47 expressed a similar view "*I feel like the final two videos could've been combined together."* It could be noted however, that despite some participants stating that they already understood the animation videos, their relatively low scores on the posttest suggest otherwise, which is in line with the fact that students do not always correctly evaluate their own knowledge (Aleven & Koedinger, 2000).

An additional reason is participants' age. Much of the prior research on concreteness fading was done with children. Younger individuals might benefit more than adults from explicit repetition. More work is needed to better understand where the line lies between enough explanation and so much repetition that students lose focus. This is a difficult balance to strike because it also depends on individual differences so it may be hard to find one method that will fit all.

Another difference between the current study and prior work is in the domain. As described in Chapter 2, the majority of research on concreteness fading is in the mathematics domain. The few studies that are in the programming domain, were included in this thesis. One being Arawjo et al. (2017), who also found no significant effect of condition. Yet their work was a game rather than an instructional lesson, so the different context does not transfer entirely to this present work. The other work being Suh et al. (2021), who are still developing their study and hence do not have experimental data yet.

Programming and mathematics certainly share similarities; for instance, they both involve logical thinking applied to problem solving. In programming, as in mathematics, there is specific notation and rules that must be followed that render some concepts abstract since they are removed from concrete reality. But perhaps the notion of abstraction in programming differs from that of mathematics, which is an interesting topic of research on its own. This may in turn affect the efficacy of concreteness fading depending on the domain of instruction.

In addition, participants with the highest learning gains were from Health Sciences, Biology, Engineering, Law, Cognitive Science, Education, and Psychology majors. Future work could further explore the connection between Science majors and their performance in the programming domain, because in this study, over 40% of participants were from Social Science.

Yet another reason for the lack of a difference could be due to experimental power. An accurate *a priori* power analysis requires knowing the population effect size, but there is very little prior work on concreteness fading in the programming domain, meaning this information is not available. Our study had sufficient power in the presence of a large population effect (d = 0.8), namely the standard statistical power of 79%, but was underpowered if the population effect was medium, d = 0.5 (in which case we would have needed 61 participants per condition, or 122 in total, to achieve 80% statistical power). In our study, the sample effect size related to the comparison between the experimental groups was very small. Thus, even if a difference could be identified by increasing the sample size, there would be little practical benefit of the concreteness fading paradigm if this sample effect size held in a subsequent study.

7.3 Limitations and Future Work

More work is needed refining the animations for the concreteness fading lesson. Although participant feedback in section 6.2 showed that seeing a concrete instantiation as an animation before seeing the abstract code was helpful, participants did also mention that some parts were repetitive. Reducing the length of the animations by removing duplicate explanations may also help with the issue of participants losing focus.

In future work, additional experiments could be conducted using a different analogy in the concrete stage. Even though efforts were spent on ensuring a solid mapping from the chosen analogy to the model of a for-loop (as explained in Chapter 3 and shown in Table 3.1), it is possible that a more appropriate analogy exists.

It is also possible that the final abstract stage (code) is already 'concrete' in the sense that it uses variables (such as 'shelf', 'book_label') that are already familiar to participants. This may confound with the results if using such a concrete example is sufficient for participants to understand the code. The second activity in the Tutor and several questions on the pre- and posttest contained variables with random names (not related to real-world scenarios), to ensure participants understand the program execution correctly instead of guessing based on context given by the variable names. However, the lesson itself (with the animations), did not include such random variable names. More work would be required to discern if this point in particular had an effect on learning.

In addition, the posttest only captured immediate learning. However, newly-acquired information may require time to consolidate and assimilate with existing knowledge. To account for this, a delayed posttest could be used. Some of the prior research on concreteness fading has used delayed posttests to measure learning (including Kaminski

et al., 2008a, 2008b; McNeil & Fyfe 2012), as it allows researchers to see if knowledge has been retained and/or changed over time. However, delayed posttests are not always feasible to implement (some people cannot be reached or no longer wish to do the delayed posttest when the time comes). Delayed posttests are much easier in classroom settings, which is what many of the concreteness fading studies did (Fyfe et al., 2015; McNeil & Fyfe 2012).

Another limitation is the context. Rather than an authentic classroom study, the experiment was conducted in an online setting. This may have affected participants' level of engagement, especially since their cameras were off.

Finally, as noted above, one reason why we found no effect of condition on learning gains could be because our sample size was not big enough. In general, NHST does not allow us to draw conclusions about null results (Aberson, 2002), something that Bayesian statistics can remedy through a framework that includes evidence for both the null and alternative model.

Data mining can help provide insight into the process of problems solving with the Tutor to shed light on learning outcomes. DataShop has logged all participant interactions with the Tutor (see end of Section 4.2.3). Some examples for future work in this area include analyzing participant behavior on specific prompts and exercises in the lesson. This analysis could include: (1) time spent on each tutor component, (2) correctness of responses, (3) behaviors related to obtaining the answer prematurely, (4) attempts at understanding mistakes versus rushing through exercises, (5) references to animation videos during the prompts and activities, as well as correlations between these components and any conditional differences. For instance, anecdotally, sometimes when participants answered the animation prompts correctly, they still struggled with the activities. For other

participants, the reverse was true (even when they did not get the prompts correct, they still answered the activities correctly). Does this mean that the animations may have been confusing to them?

In addition, future work could expand the use of concreteness fading to other programming topics, such as 'classes', functions, or algorithms for instance. There already exists research on using simulations and animations for teaching various algorithms (Baecker, 1998), however concreteness fading was not applied in these works. Meaning, algorithmic visualizations were used in instruction, but no explicit links were made between the animation and the code. Others have also gamified the use of visualizations in programming education to increase student engagement (Yohannis & Prabowo, 2015).

7.4 Conclusion

The present thesis investigated the effectiveness of concreteness fading in the programming domain, specifically teaching the concept of for-loops through a lesson embedded in an online computer tutor. Although overall learning increased, we did not find an effect of condition on learning gains. Thus, more research is needed in applying the concreteness fading paradigm to programming education. Programming is a subject that students generally struggle with, therefore finding new pedagogical tools to help close knowledge gaps and support student learning is very much necessary.

Appendices

Appendix A

A.1 Stage 1 (Concrete) Narration Script in the Animation Video

"Imagine the following scenario:

1. On the left we see a robot (in the form of a robotic arm). At the top, we see a shelf with books on it. Each book contains a label consisting of a single letter written on it. The robot can only see things from up close, so it needs to use a magnifying glass to see the label because it is too small otherwise.

   Let's say we want to give the robot instructions to count the number of books in this shelf that are labelled with the letter E. Before it starts, this 'count' is zero, because so far it has not seen any books labelled E.
   a. To accomplish this task, the robot will need to use the magnifying glass to look at each book one by one in order.
   b. For each book it looks at, it will go to its worktable, shown below the shelf, and follow the instructions on the clipboard.
   c. The instructions tell the robot to count each time it comes across a book with the letter E. This is important: the robot cannot remember instructions like humans, so it forgets them right after it follows them – so it has to come back to this work table each time it sees a new book on the shelf.

2. The first book has the letter W on it.
   a. Remember, the magnifying glass is here because the robot can only see the letters on the books one at a time, and it uses the magnifying glass to look at each letter separately. Whichever letter the magnifying glass is on, is the letter that the robot is currently looking at. It is the only letter it can currently see.
3. While humans can immediately tell that the current letter is not E, the robot does not know what to do until it reads its instructions. So, the robot moves to the worktable and checks to see what it should do by reading the instructions on the clipboard.
4. The instructions say "count books with letter E", in other words, "if the book is labelled with the letter E, add 1 to the count".
5. The robot checks what the magnifying glass is showing for the first book. The letter on the first book is W.
6. Since the instructions say to only count books with the letter E, and the current book has the letter W and not E, the robot does not add 1 to the count.

7. So far the robot has only dealt with the first book – so it is not finished with the task (of counting all the books on the shelf that are labelled with the letter E).
8. So, the robot goes back to the shelf and moves the magnifying glass to the next book.
    a. To figure out what to do next, it goes back to the instructions on the clipboard.
    b. You may wonder why the robot does not remember those instructions from the last round. As we already said, it cannot remember instructions from before – so it has to keep coming back to its work table.
    c. The instructions say to "add 1 to count if the book is labelled with the letter E".
    d. The robot checks the magnifying glass. Since the current book has the letter E on it, the robot adds 1 to count on the clipboard. This increases the value of count from zero to 1.
9. It will keep repeating these steps kind of like a loop – move to the next book on the shelf, check the instructions on the clipboard, check what the magnifying glass shows, and add 1 to count if the instructions are satisfied, until it has gone through all the books. It will then stop."

A.2 Stage 2 (Semiconcrete) Narration Script in the Animation Video

This script is similar to Stage 1 (Concrete), but includes introduction of the code elements.

"Now you will see a second animation. This animation is similar to the previous one, but perceptual features like colour have been removed, so that it's simpler. Also some of the elements are now labeled/marked/ so we can slowly transition to code. The high-level idea in this next animation is the exact same one as in the previous animation, so there will be some repetition but that will help solidify the ideas.

1. Like before, on the left we see a robot (in the form of a robotic arm). At the top, we see a shelf with books on it. We will refer to this shelf with a variable called shelf. This is new in this animation. We use a unique name for this shelf so that we can distinguish it from other objects/shelves. Each book contains a label consisting of a single letter written on it. The label on each book is referred to as book_label. The robot needs to use a magnifying glass to see the label because it is small. 'book_label' refers to whichever letter the magnifying glass is currently looking at. **So book_label will change what it refers to, as we go through the animation.**

   Recall that we want to give the robot instructions to count the number of books in this shelf that are labelled with the letter E. We can see the instructions on the clipboard on the table. We can write these instructions in a programming language, for instance Python, as if book_label == 'E' then count = count + 1. This means: **if** the book is labelled with the letter E **then** add 1 to count. Before the robot starts, this 'count' is zero, because so far the robot has not seen any books labelled E. We can refer to this as count = 0. This is the exact same idea as in the previous animation.

   a. To accomplish this task, the robot will need to use the magnifying glass to look at each book one by one in order.
   b. **For each book it looks at**, it will go to its worktable, and follow the instructions on the clipboard.
2. The first book has the letter W on it so book_label refers to the letter W.
   a. Remember, the magnifying glass is here because the robot can only see the letters on the books one at a time, and it uses the magnifying glass to look at each letter separately. Whichever letter the magnifying glass is on, is the letter that the robot is currently looking at. It is the only letter it can currently see.
3. The robot moves to the worktable and checks to see what it should do by reading the instructions on the clipboard.
4. Do you remember what the instructions say? "if the book is labelled with the letter E, add 1 to count".
5. The robot checks which book_label the magnifying glass is showing for the first book. The letter on the first book is W.

6. Since the instructions say to only count books with the letter E, and the current book has the letter W and not E, the robot does not add 1 to the count.
7. The robot goes back to the shelf and moves the magnifying glass to the next book. Book_label also moves and it now refers to the label on the next book.
    a. To figure out what to do next, it goes back to the instructions on the clipboard.
    b. The robot checks which book_label the magnifying glass is showing. Since the current book has the letter E on it, book_label now refers to the letter E, and hence the robot adds 1 to count. 'count' will be updated each time the robot counts plus one when it sees a book that is labelled with the letter E.
8. The robot then goes back to the shelf and moves the magnifying glass to the next book. Since the next book has the letter R on it, book_label now refers to the letter R.
9. It will keep repeating these steps kind of like a loop – move to the next book on the shelf, check the instructions on the clipboard, check what the magnifying glass shows, and add 1 to count if the instructions are satisfied, until it has gone through all the books. It will then stop."

A.3 Stage 3 (Abstract) Narration Script in the Animation Video

"We are now ready to look at a python program that counts the number of letter Es in a given list. The program is shown in the image on the screen. We will make connections between this program and the previous animations we saw. Recall that a program is a series of instructions called code that we give to a computer telling it how to perform a task. Let's trace this program to go through the same series of steps that the computer would take when executing this program.

In the previous animations, the robot arm played the role of the computer going through this current program.

1. In the first line of the program, we have a variable named shelf. It is now highlighted in the program to bring attention to it. Assigned to this shelf variable is a list containing the following strings ['W', 'E', 'R', 'E', 'E', 'T'].

If we compare this code to the previous animations we saw, the equivalent of the list containing these letters, was the shelf containing books with the corresponding letters on them.
In line 3 of the program, we have a variable named count, that will keep track of the books that have the label E. In other words, count keeps track of strings E. In line 3, the program assigns count the value of 0.
Why zero? Because at this point in the program, it has not seen any books with the letter E).

   a. To accomplish the task of counting how many letter Es are in the list, the program will use a for loop. Here is a high level description of how it works:
   b. A for loop looks at each letter in the list, the one right after the word "in", one by one in order. Even though we as humans can quickly see all the letters in the list, the computer can only see them one by one in order. For a given letter, it assigns that letter to the variable called book_label. **The value of this variable will change as the program runs**.
   c. Remember in the previous animations, the magnifying glass symbolized the idea that the computer can only look at one letter at a time, and each time the magnifying glass moved to the next book, book_label also changed what it referred to. It is the same in this case with the code we see.
   d. Once it assigns the current letter to book_label, the computer enters the body of the for loop – the indented part of the program - and executes the instructions inside.
      These are like the instructions on the clipboard we saw in the previous two animations. The idea is the same.
   e. Line 6 checks if the current letter, stored in book_label, is an E and if so, adds 1 to the variable count.
      This is important: the computer can't remember instructions like humans so it forgets them right after it follows them – so it has to come back to this part in the loop each time it sees a new letter in the list.

2. The first letter in the list is W – the program assigns this to the book_label variable. book_label now corresponds to the string W.

Compared to the previous animations we saw, the equivalent of the letter W in this list was a book on the shelf, with the letter W on the book.

3. While humans can immediately tell this letter is not E, the computer doesn't know what to do until it reads its instructions. So, the computer goes inside the body of the loop and checks to see what it should do by reading the instructions written there.
   Comparing this to the previous animations, the equivalent of the computer going into the loop, was the robot arm going to the clipboard on its worktable, and then looking at the instructions on the clipboard.

4. The instructions say that "if book_label is the letter E, add 1 to count". Remember that book_label refers to whichever letter the computer is currently looking at.
5. The computer is currently looking at the first letter in the list and so book_label corresponds to W.
6. Since the instructions say to only add 1 to count if book_label has the value E, and since the current value of book_label is W and not E, the computer does not add 1 to count.
7. The computer will keep repeating these steps – assign the current letter to the variable book_label, move inside the loop, check if book_label is equal to the letter E, add 1 to count if so, and print the current count, until it has gone through all the letters in the list. Once the loop has gone through all the elements in the list – in this case, all the string letters in the list, the computer moves onto the next instruction right after the loop. If there are no instructions after the loop, the program ends. In this case, the final instructions say to print "We are done!"
8. So far the computer has only dealt with the first book – so it is not finished with the task (of counting all the letter Es in the list).
9. So, the computer goes back to the list and moves on to the next letter.
   a. It assigns that letter to book_label.
   b. To figure out what to do next, it goes to the instructions inside the loop.
   c. The instructions say that "if book_label is the letter E, add 1 to count".
   d. The computer checks which letter it is currently looking at in the list. Since it is currently looking at the letter E, the program adds 1 to count. Count now equals 1.

   If we print count now, the number 1 will be printed, since count currently refers to 1. On the right we can see what will be printed in the output shell in Python. 'count' will increase by 1 each time the computer finds the letter E in shelf.
   Comparing this code to the previous animation, the equivalent of count = count + 1, print(count) was when the robot updated the count on the clipboard. The print(count) statement is like the robot writing the count on the clipboard and showing us so we can see what it is.

10. The computer now moves on to the next letter in the list.

The next letter is R, so book_label refers to R. The computer enters the body of the loop, checks if book_label is the letter E, since it is not, it moves to the next letter in the list.
The next letter is E, so book_label now refers to E. The computer enters the body of the loop, checks if book_label is the letter E, since book_label **is** the letter E, the computer adds one to count, then prints count.

The computer moves to the next letter in the list which is E, so book_label refers to E. The computer enters the loop body, checks if book_label is the letter E, since book_label **is** the letter E, the computer adds one to count, then prints count. Count is now 3. The computer moves to the last letter in the list, which is T. So book_label now refers to T. The computer enters the body of the loop, checks if book_label is the letter E, since it is not, count is not updated.

Since the computer has now gone through all the elements in the list, it now moves to the instructions outside of the loop, and prints “We are done!”)
The program is now over.”

Appendix B

B.1 Prompts in the For-Loop Lesson in the Computer Tutor (Answers in Italic)

CONCRETENESS FADING CONDITION

STAGE 1 Prompts:

1. What are the three steps that repeat again and again? In your answer, indicate the location (bookshelf, table, notebook) and what happens in that location.
The pattern is (1) Bookshelf: check name of current book; (2) Table: check instructions in notebook; (3) Notebook: update count if condition is met;

2. How many times does the robot visit the notebook to check the instructions?
The robot checks the instructions as many times as there are books on the shelf.

STAGE 2 Prompts:

1. The variable book_label corresponds to the magnifying glass in the first animation, and how it can only see one book at a time. Describe how the value of book_label changes as the program runs.
It is assigned the value of the current book label while the loop runs. Book_label moves/changes when the robot moves to the next book.

2. Describe how the value of count changes as the program runs.
The count variable is increased by 1 each time a book with the letter "E" is encountered in the list of strings stored in shelf.

3. Name one similarity and one difference between the instructions in the notebook ("count books with letter E") and the instructions written above the notebook ("if book_label == 'E', count = count + 1")? Focus on a higher level answer, rather than focusing on how the text looks.
Similarity – they mean the same thing. Difference – the instructions on the notebook are written in English, and the other ones are written in a programming language (Python).

STAGE 3 Prompts:

1. The computer executing the program is symbolized by what in the previous animation(s)?
The robot arm.

2. The book shelf in the first two animations was an analogy for which variable in the program in this third animation? *The shelf holding all the books – just like a book shelf in your home. The list 'shelf'.*

3. Look at the code in this video. Which part of the code corresponds to the robot arm moving to the instructions in the notebook in the previous animations?
Entering the indented part of the code, inside the for loop (ie. the body of the loop).

4. In the previous animations, what did the magnifying glass symbolize, in terms of the code in this current video?
The magnifying glass symbolized the idea that the computer can only look at one letter at a time, and each time the magnifying glass moved to the next book book_label also changed what it referred to.

NEXT screen (present in ABSTRACT-ONLY condition as well):

5. In this code we saw a bookshelf. Can you think of a real world example of a loop where you go through each element of a sequence, that does not involve a bookshelf?
Many things – a line of people at startbucks could be a list; a sequence of houses on a street could be a list; a series of assignment scores is also a list. A list is a sequence of elements that can be strings, numbers etc.
A loop would be to go through each element in the sequence/list and do something depending on the instructions. So for example, we could count the number of people in a starbucks line that want to order coffee vs. tea, place a mark on each odd-numbered house on a street, count the number of assignment scores that are greater than 60, etc.

6. What are the three steps the computer keeps repeating as part of the for loop?
Assign the current letter to the variable book_label; Move inside the loop; Check if the book_label is equal to E.

7. How does the computer know what instructions are in the body of the for loop?
The indentation –all the code in the loop is indented.

8. When does the loop end?
When all the elements in the list have been checked.

9. The for loop syntax for the 1st line we saw in this program was:
for book_label in shelf:
Could book_label have a different name – like var or x or bananas? Explain.
Yes – this is a variable and as we saw in the tutorial the name of the variable is up to the programmer.

ABSTRACT-ONLY CONDITION

STAGE 3 Prompts:

1. What are the three steps that repeat again and again? In your answer, indicate the location (shelf, instructions, count) and what happens in that location.
The pattern is (1) shelf: check name of current book; (2) loop body: check instructions in loop body; (3) count: update count if condition is met.

2. What role does the computer play when executing this program?
It goes through each line one by one and executes it.

3. What does the variable shelf represent?
It is a list containing different strings (that represent book labels). The shelf holds all the books, just like a bookshelf in your home.

4. Describe the role of book_label in the code above, and how its value changes as the program runs.
Book_label symbolizes the idea that the computer can only look at one letter at a time, It is assigned the value of the current book label while the loop runs. Book_label moves/changes when the computer moves to the next element of the list.

5. Describe how the value of count changes as the program runs.
The count variable is increased by 1 each time a book with the letter "E" is encountered in the list of strings stored in shelf.

NEXT screen:

6. In this code we saw a bookshelf. Can you think of a real world example of a loop where you go through each element of a sequence, that does not involve a bookshelf?
Many things – a line of people at startbucks could be a list; a sequence of houses on a street could be a list; a series of assignment scores is also a list. A list is a sequence of elements that can be strings, numbers etc.
A loop would be to go through each element in the sequence/list and do something depending on the instructions. So for example, we could count the number of people in a starbucks line that want to order coffee vs. tea, place a mark on each odd-numbered house on a street, count the number of assignment scores that are greater than 60, etc.

7. What are the three steps the computer keeps repeating as part of the for loop?
Assign the current letter to the variable book_label; Move inside the loop; Check if the book_label is equal to E.

8. How does the computer know what instructions are in the body of the for loop?
The indentation –all the code in the loop is indented.

9. When does the loop end?
When all the elements in the list have been checked.

10. The for loop syntax for the 1st line we saw in this program was:
for book_label in shelf:
Could book_label have a different name – like var or x or bananas? Explain.
Yes – this is a variable and as we saw in the tutorial the name of the variable is up to the programmer.

Appendix C

C.1 Demographics Questionnaire

The demographics questionnaire contained the following four open-ended questions:

What is your age?

What is your gender (leave blank if you do not wish to answer)?

What is your major?

List the programming classes you have taken (leave blank if none)?

Appendix D

D.1 Pretest

Questions on the pretest. The pretest in the study was administered online through Google Forms.

#1
What does the program below print?

```
result = 1
result = result + 10
print(result)
```

#2
What does the program print?

```
result = "orange"
result = result + "x"
print(result)
```

#3
What does the program print?

```
answer = "Bob"
print("Ann")
if answer == "Ann":
    print("hello")
print("hi")
```

#4
What does the program print?

```
counter = 1
if counter < 2:
    counter = counter + 2
if counter > 2:
    counter = counter + 1
print(counter)
```

#5
What does the program print?

```
var = "bear"
count = 2
if var == "wolf":
    print("hi", var)
    count = count + 1
print(var, count)
```

#6
What does the program print?

```
stuff = [800, 40, 100, 150, 0]
var = 1
for x in stuff:
    if x > 50:
        var = var + 1
print(var)
```

#7
Describe in general terms what the program in Question 6 above does?

#8
What does the program print?

```
things = ["x", "y", "x", "z", "t", "a"]
var = 10
for ele in things:
    if ele == "x":
        var = var - 1
print(var)
```

#9
What does the program print?

```
var1 = ["number", "flower", "key"]
temp = "-"
for element in var1:
    temp = element + temp
    print("hi")
print(temp)
```

#10
What does the program print?

```
list2 = [7, 5, 3, 9, 4, 15, 2]
var = 0
res = 20
for ele in list2:
    if res < 6:
        res = res - ele
    var = var + 1
print(res, var)
```

#11
What does the program print?

```
val = 10
l1 = [30, 10, 20]
found = "False"
for ele in l1:
    if ele == 10:
        found = "True"
    if not ele == 10:
        found = "False"
print(found)
```

#12
Write a program in Python following these instructions:
Start with the following list:
bugs = ["bug1", "bug0", "bug54", "bug786", "bug0", "bug23"]
Go through this list and print all the list elements, one per line, and check whether the list contains the string "bug0".
If it does, print "yes" - it should only print yes once, even if "bug0" shows up more than once.

D.2 Posttest

Questions on the posttest. The posttest in the study was administered online through Google Forms.

#1
What does the program below print?

```
res = 1
res = res + 20
print(res)
```

#2
What does the program print?

```
res = "apple"
res = res + "z"
print(res)
```

#3
What does the program print?

```
answer = "Ann"
print("Bob")
if answer == "Bob":
    print("hi")
print("hello")
```

#4
What does the program print?

```
counter = 1
if counter < 2:
    counter = counter + 2
if counter > 2:
    counter = counter + 1
print(counter)
```

#5
What does the program print?

```
var = "bird"
count = 3
if var == "cat":
    print("hi", var)
    count = count + 1
print(var, count)
```

#6
What does the program print?

```
stuff = [800, 40, 100, 150, 0]
var = 1
for x in stuff:
    if x > 50:
        var = var + 1
print(var)
```

#7
Describe in general terms what the program in Question 6 above does?

#8
What does the program print?

```
things = ["x", "y", "x", "z", "t", "a"]
var = 5
for ele in things:
    if ele == "x":
        var = var - 1
print(var)
```

#9
What does the program print?

```
var1 = ["number", "flower", "key"]
temp = "-"
for element in var1:
    temp = element + temp
    print("hi")
print(temp)
```

#10
What does the program print?

```
list2 = [7, 5, 3, 9, 4, 15, 2]
var = 0
res = 30
for ele in list2:
    if res < 6:
        res = res - ele
    var = var + 1
print(res, var)
```

#11
What does the program print?

```
val = 10
l1 = [30, 10, 20]
found = "False"
for ele in l1:
    if ele == 10:
        found = "True"
    if not ele == 10:
        found = "False"
print(found)
```

#12
Write a program in Python following these instructions:
Start with the following list:
trees = ["0tree", "5tree", "89tree", "376tree", "5tree", "23tree"]
Go through this list and print all the list elements, one per line, and check whether the list contains the string "5tree".
If it does, print "yes" - it should only print yes once, even if "5tree" shows up more than once.

Appendix E

E.1 Consent Form

Informed Consent Form

Name and Contact Information of Researchers:

Kasia Muldner, Carleton University, Department of Cognitive Science
Email: Kasia.Muldner@carleton.ca
Nadia Markova, Carleton University, Department of Cognitive Science
Email: nadiamarkova@cmail.carleton.ca

Project Title

Student learning & Educational technologies

Project Sponsor and Funder (if any)

NSERC Discovery grant

Carleton University Project Clearance

This study has received clearance by the Carleton University Research Ethics Board B (Clearance #**111476**).

Invitation

You are invited to participate in our online research study. This study involves filling in a brief questionnaire about programming, watching a brief tutorial and then using several online interfaces to predict the output of computer programs. To be eligible you cannot have taken more than one university-level class.

The information in this form is intended to help you understand what we are asking of you so that you can decide whether you agree to participate in this study. Your participation in this study is voluntary, and a decision not to participate will not be used against you in any way. As you read this form, and decide whether to participate, please ask all the questions you might have, take whatever time you need, and consult with others as you wish.

What is the purpose of the study?

The purpose of this study is to analyze how you interact with different versions of an interface that teaches you how to predict the output of programs.

What will I be asked to do?
The study is conducted online using the Zoom platform. If you agree to take part in the study, we will ask you to fill out some questionnaires on programming – you are not expected to have much background and it's okay if you don't know the answers to it. You will work virtually and the researcher will be online with you during the session. We will ask you to use several interfaces that teach about computer programming. We will use Zoom to audio record the session and the screen, including your actions in the interface (your face will not be recorded). This in-session data will be stored locally on the researcher's computer. Operation data, such as meeting and performance data, will be stored and protected by Zoom on servers located in North America, but may be disclosed via a court order or data breach. The audio and screen-recording data will NOT be shared or publicly used and will be destroyed as soon as it is transcribed and analyzed. Please note that audio and screen recording is part of the data collection for this study, and so is a condition of consenting to participate After the instructional activity, you will fill out a questionnaire similar to the one from the beginning. The study will take no more than 2 hours.

Risks and Inconveniences
We do not anticipate any risks to participating in this study.

Possible Benefits
You may not receive any direct benefit from your participation in this study. However, your participation may allow researchers to better understand how to design interfaces to support student learning of programming.

Compensation/Incentives
As a token of appreciation, you will receive 2% extra credits for CGSC 1001 OR $25 [if not enrolled in CGSC1001]. If you withdraw before finishing the experiment, the amount of compensation will be prorated by time (e.g. if you withdraw halfway through the study, you will receive half the compensation).

No waiver of your rights
By signing this form, you are not waiving any rights or releasing the researchers from any liability.

Withdrawing from the study
During the study session, you have the right to end your participation for any reason, by stating that you do not want to continue. If you withdraw from the study, all information you have provided will be immediately destroyed (including the audio recording). Because we do not store any identifying information with the data, once you leave the session, withdrawal is not possible.

Confidentiality
We will treat your personal information as confidential, although absolute privacy cannot be guaranteed. No information that discloses your identity will be released or published.

Research records may be accessed by the Carleton University Research Ethics Board in order to ensure continuing ethics compliance.
You will be assigned a code so that your identity will not be directly associated with the data you have provided. However, if you are being granted course credit for taking part in the study, identifying information will be retained by the SONA system using a code until the course credit is granted.
We will password protect any research data that we store or transfer. Research data will only be accessible by the researchers and the research supervisor and will not have any identifiable information. Thus, your name will not appear in any publications or other venues. Once the project is completed, research data will be kept and potentially used for other research projects on this same topic (but as noted above, audio and research data will be destroyed as soon as it is transcribed/analyzed).

Data Retention
After the study is completed, your de-identified data will be retained for future research use (note that the audio and screen recordings will be destroyed once they are transcribed by the research team).

New information during the study
In the event that any changes could affect your decision to continue participating in this study, you will be promptly informed.

Ethics review
This project was reviewed and cleared by the Carleton University Research Ethics Board B. If you have any ethical concerns with the study, please contact Carleton University Research Ethics Board (by phone at 613-520-2600 ext. 4085 CUREB B or by email at ethics@carleton.ca).

Statement of consent – print name
I voluntarily agree to participate in this study. ___Yes ___No

Signature of participant (typed name) Date

_____ _____

Research team member who interacted with the participant
I have explained the study to the participant and answered any and all of their questions. The participant appeared to understand and agree. I offered to send a copy of the consent form to the participant for their reference.

_____ _____
Signature of researcher (typed name) Date

References

Aberson, C. (2002). Interpreting Null Results: Improving Presentation and Conclusions with Confidence Intervals. *Journal of Articles in Support of the Null Hypothesis, JASNH*, *1*(3), 36–42.

Aleven, V., & Koedinger, K. R. (2000). Limitations of Student Control: Do Students Know when They Need Help? In G. Gauthier, C. Frasson, & K. VanLehn (Eds.), *Intelligent Tutoring Systems* (Vol. 1839, pp. 292–303). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-45108-0_33

Aleven, V., McLaren, B. M., Sewall, J., van Velsen, M., Popescu, O., Demi, S., Ringenberg, M., & Koedinger, K. R. (2016). Example-Tracing Tutors: Intelligent Tutor Development for Non-programmers. *International Journal of Artificial Intelligence in Education*,*26*(1), 224–269.https://doi.org/10.1007/s40593-015-0088-2

Al-Sakkaf, A., Omar, M., & Ahmad, M. (2019). A systematic literature review of student engagement in software visualization: A theoretical perspective. *Computer Science Education*, *29*(2–3), 283–309. https://doi.org/10.1080/08993408.2018.1564611

Arawjo, I., Wang, C.-Y., Myers, A. C., Andersen, E., & Guimbretière, F. (2017). Teaching Programming with Gamified Semantics. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 4911–4923. https://doi.org/10.1145/3025453.3025711

Baecker, R. (1998). Sorting Out Sorting: A Case Study of Software Visualization for Teaching Computer Science. *Software Visualization: Programming as a Multimedia Experience.* MIT Press, 369-381.

Bandura, A. (1994). Self-efficacy. In V. S. Ramachaudran (Ed.), *Encyclopedia of human behavior* (Vol. 4, pp. 71-81). New York: Academic Press. (Reprinted in H. Friedman [Ed.], *Encyclopedia of mental health*. San Diego: Academic Press, 1998).

Braithwaite, D. W., & Goldstone, R. L. (2013). Integrating formal and grounded representations in combinatorics learning. *Journal of Educational Psychology*, *105*(3), 666–682. https://doi.org/10.1037/a0032095

Bronkhorst, H., Roorda, G., Suhre, C., & Goedhart, M. (2021). Student Development in Logical Reasoning: Results of an Intervention Guiding Students Through Different Modes of Visual and Formal Representation. *Canadian Journal of Science, Mathematics and Technology Education*, *21*(2), 378–399. https://doi.org/10.1007/s42330-021-00148-4

Bruner, J. (1966). *Toward a Theory of Instruction.* Cambridge, MA: Harvard University Press.

Cognitive Development. Idea to Form. (2016, October 31). Retrieved from https://idea2form.wordpress.com/2016/10/31/cognitive-development/

De Bock, D., Deprez, J., Dooren, W. V., Roelens, M., & Verschaffel, L. (2011). Abstract or Concrete Examples in Learning Mathematics? A Replication and Elaboration of Kaminski, Sloutsky, and Heckler's Study. *Journal for Research in Mathematics Education*, *42*(2), 109–126. https://doi.org/10.5951/jresematheduc.42.2.0109

Dixon, R. A., & Brown, R. A. (2012). Transfer of Learning: Connecting Concepts During Problem Solving. *Journal of Technology Education*, *24*(1). https://doi.org/10.21061/jte.v24i1.a.1

Dweck, C. S. (2008). Mindsets and Math/Science Achievement. *Prepared for the Carnegie Corporation of New York-Institute for Advanced Study Commission on Mathematics and Science Education.*

Esponda-Argüero, M. (2008). Algorithmic Animation in Education—Review of Academic Experience. *Journal of Educational Computing Research*, *39*(1), 1–15. https://doi.org/10.2190/EC.39.1.a

Fyfe, E. R., McNeil, N. M., & Borjas, S. (2015). Benefits of "concreteness fading" for children's mathematics understanding. *Learning and Instruction*, *35*, 104–120. https://doi.org/10.1016/j.learninstruc.2014.10.004

Fyfe, E. R., McNeil, N. M., Son, J. Y., & Goldstone, R. L. (2014). Concreteness Fading in Mathematics and Science Instruction: A Systematic Review. *Educational Psychology Review*, *26*(1), 9–25. https://doi.org/10.1007/s10648-014-9249-3

Fyfe, E. R., & Nathan, M. J. (2019). Making "concreteness fading" more concrete as a theory of instruction for promoting transfer. *Educational Review*, *71*(4), 403–422. https://doi.org/10.1080/00131911.2018.1424116

Gentner, D. (1983). Structure-Mapping: A Theoretical Framework for Analogy*. *Cognitive Science*, *7*(2), 155–170. https://doi.org/10.1207/s15516709cog0702_3

Gentner, D., Brem, S., Ferguson, R. W., Markman, A. B., Levidow, B. B., Wolff, P., & Forbus, K. D. (1997). Analogical Reasoning and Conceptual Change: A Case Study of Johannes Kepler. *Journal of the Learning Sciences*, *6*(1), 3–40. https://doi.org/10.1207/s15327809jls0601_2

Gentner, D., & Holyoak, K. J. (1997). Reasoning and Learning by Analogy. *American Psychological Association 52*(1)*,* 32-34.

Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. *Cognitive Psychology*, *15*(1), 1–38. https://doi.org/10.1016/0010-0285(83)90002-6

Goldstone, R. L., & Son, J. Y. (2005). The Transfer of Scientific Principles Using Concrete and Idealized Simulations. *Journal of the Learning Sciences*, *14*(1), 69–110. https://doi.org/10.1207/s15327809jls1401_4

Harnad, S. (1990). THE SYMBOL GROUNDING PROBLEM. *Physica D*, 42, 335-346.

Jaakkola, T., & Veermans, K. (2020). Learning electric circuit principles in a simulation environment with a single representation versus "concreteness fading" through multiple representations. *Computers & Education*, *148*, 103811. https://doi.org/10.1016/j.compedu.2020.103811

Johnson-Laird, P.N., Girotto, V., & Legrenzi, P. (1998). Mental models: a gentle guide for outsiders.

Kaminski, J. A., Sloutsky, V. M., & Heckler, A. F. (2008a). The Advantage of Abstract Examples in Learning Math. *Science*, *320*(5875), 454–455. https://doi.org/10.1126/science.1154659

Kaminski, J. A., Sloutsky, V. M., & Heckler, A. F. (2008b). Supporting Online Material for: The Advantage of Abstract Examples in Learning Math. *Science*, *320*(5875), 454–455. https://doi.org/10.1126/science.1154659

Kim, H. (2020). Concreteness Fading Strategy: A Promising and Sustainable Instructional Model in Mathematics Classrooms. *Sustainability*, *12*(6), 2211. https://doi.org/10.3390/su12062211

Koedinger, K.R., Baker, R.S.J.d., Cunningham, K., Skogsholm, A., Leber, B., Stamper, J. (2010). A Data Repository for the EDM community: The PSLC DataShop. In

Romero, C., Ventura, S., Pechenizkiy, M., Baker, R.S.J.d. (Eds.) *Handbook of Educational Data Mining*. Boca Raton, FL: CRC Press.

Krishnamurthi, S., & Fisler, K. (2019). Programming Paradigms and Beyond. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (1st ed., pp. 377–413). Cambridge University Press. https://doi.org/10.1017/9781108654555.014

Laski, E. V., Jor'dan, J. R., Daoust, C., & Murray, A. K. (2015). What Makes Mathematics Manipulatives Effective? Lessons From Cognitive Science and Montessori Education. *SAGE Open*, *5*(2), 215824401558958. https://doi.org/10.1177/2158244015589588

Laski, E. V., & Siegler, R. S. (2014). Learning from number board games: You learn what you encode. *Developmental Psychology*, *50*(3), 853–864. https://doi.org/10.1037/a0034321

McNeil, N. M., & Fyfe, E. R. (2012). "Concreteness fading" promotes transfer of mathematical knowledge. *Learning and Instruction*, *22*(6), 440–448. https://doi.org/10.1016/j.learninstruc.2012.05.001

Miller, S. P., & Hudson, P. J. (2007). Using Evidence-Based Practices to Build Mathematics Competence Related to Conceptual, Procedural, and Declarative Knowledge. *Learning Disabilities Research & Practice*, *22*(1), 47–57. https://doi.org/10.1111/j.1540-5826.2007.00230.x

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, *13*(2), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computing Education*, *13*(2), 1–31. https://doi.org/10.1145/2483710.2483713

Stacey, K., Helme, S., Archer, S., & Condon, C. (2001). THE EFFECT OF EPISTEMIC FIDELITY AND ACCESSIBILITY ON TEACHING WITH PHYSICAL MATERIALS: A COMPARISON OF TWO MODELS FOR TEACHING DECIMAL NUMERATION. *Educational Studies in Mathematics, 47*, 199-221.

Suh, S., Latulipe, C., Lee, K. J., Cheng, B., & Law, E. (2021). Using Comics to Introduce and Reinforce Programming Concepts in CS1. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 369–375. https://doi.org/10.1145/3408877.3432465

Suh, S., Lee, M., Xia, G., & Law, E. (2020). Coding Strip: A Pedagogical Tool for Teaching and Learning Programming Concepts through Comics. *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 1–10. https://doi.org/10.1109/VL/HCC50065.2020.9127262

Trninic, D., Kapur, M., & Sinha, T. (2020). The Disappearing "Advantages of Abstract Examples in Learning Math". *Cognitive Science, 44*(7), e12851. https://doi.org/10.1111/cogs.12851

Trory, A., Howland, K., & Good, J. (2018). Designing for concreteness fading in primary computing. *Proceedings of the 17th ACM Conference on Interaction Design and Children*, 278–288. https://doi.org/10.1145/3202185.3202748

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., Tan, A. H., Hwa, L., Li, M., & Ko, A. J. (2019). A theory of instruction for introductory programming

skills. *Computer Science Education*, *29*(2–3), 205–253. https://doi.org/10.1080/08993408.2019.1565235

Yohannis, A., & Prabowo, Y. (2015). Sort Attack: Visualization and Gamification of Sorting Algorithm Learning. *2015 7th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games)*, 1–8. https://doi.org/10.1109/VS-GAMES.2015.7295785