A COMPUTER LEARNING ENVIRONMENT FOR NOVICE JAVA PROGRAMMERS THAT

SUPPORTS COGNITIVE LOAD REDUCING ADAPTATIONS AND

DYNAMIC VISUALIZATIONS OF COMPUTER MEMORY

by

James Stephen Williams

A Dissertation Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

in Engineering

at

The University of Wisconsin-Milwaukee

August 2014

ABSTRACT
A COMPUTER LEARNING ENVIRONMENT FOR NOVICE JAVA PROGRAMMERS THAT
SUPPORTS COGNITIVE LOAD REDUCING ADAPTATIONS AND
DYNAMIC VISUALIZATIONS OF COMPUTER MEMORY

by

James Stephen Williams

The University of Wisconsin-Milwaukee, 2014
Under the Supervision of Professor Susan McRoy

Learning to program a computer is difficult for many. The Learning Edge Momentum

hypothesis suggests that the difficulty may be due to the tightly integrated nature of

programming concepts and adapting the way curriculum is offered may have a

significant influence on the outcomes. We investigate applying cognitive load reducing

methods to instruction of the introductory programming concepts of declaration,

assignment and sequence, using a new learning environment that an instructor can

adapt for a specific example or that a student can personalize for amount and modality

of content provided. Our study has three learning surveys. Each learning survey has

short instructional videos designed using cognitive load reducing methods and then asks

participants to solve novel problems using the presented materials. Our first learning

survey was completed by 123 participants recruited on Amazon's Mechanical Turk

(AMT). We found that 23% that watched the instructional video without computer

memory (n=61) answered the three code tracing questions correctly. Our second

learning survey included instructional videos prepared after analyzing the results of the

previous survey and emphasized cognitive load reducing methods in preparing the new

instruction. This second survey was completed by 220 participants also recruited via

AMT. We found that 57% of the participants that watched the instructional video

without computer memory (n=72) answered the three tracing questions correctly. Our

third learning survey with 322 participants recruited via AMT confirmed that the

difference between the two videos was statistically significant with medium effect size.

In the third survey, 29% of the participants watching the first survey instructional video

without computer memory and 45% of those that watched the second survey

instructional video without computer memory answered all three tracing questions

correctly. In the third learning survey, the gain from 29% from our first short video that

we thought was a reasonable presentation to 45% in the second short video seems to

lend strong support to the hypothesis that our typical methods of instruction for

introductory programming simply overwhelm the cognitive capabilities of many of the

students. Our results suggest that cognitive load reducing methods may be very helpful

for teaching introductory programming concepts.

ACKNOWLEDGEMENTS

First, I would like to thank my primary advisor, Susan McRoy, who has provided consistent, wise council that took into account my balancing work and family with my studies. She gave me freedom to explore while encouraging me to keep narrowing down my work. My committee members Ethan Munson, Tian Zhao, Cindy Walker and Dennis Brylow provided valuable feedback and advice that guided my research and presentation.

Thank you to Joe and Bobbie Bradley along with neighbors and many friends who have been flexible and assisted with childcare.

I am grateful to my dad, Carl Williams, for getting me started with my first computer, a Sinclair ZX-81, and helping me learn to program. I am grateful to my mom, Katie Canuel, for her unwavering support with whatever I chose to do. I appreciate the support and encouragement of my brother Paul and sisters Amy and Sara. I am grateful for the support and encouragement of my late sister Annie with earlier work.

Thank you to my kids, Molly, Kevin and Erin, who have lived most of their lives with their dad going to school too.

And most of all I am grateful to my beautiful wife, Tara, who has been a wonderful partner with our kids, work and school. I can't imagine accomplishing what we have without you. I love you!

TABLE OF CONTENTS

List of Figures

List of Tables

1. Introduction

Personal Motivation and Inspiration

As an instructor, I have spent many hours introducing programming to novices and I have had the experience of seeing some students seemingly just "get it" and others that really struggled. Over the years, I have tried workshop settings, various tools such as Alice and Jeliot, group work and various kinds of assignments. Essentially, my attempts to improve programming instruction were based on my intuition and at times seemed to work for some students but not others. I found myself increasingly wanting to know what is really effective. I was familiar with John R. Anderson's ACT-R research of which a key finding is "the acquisition and performance of a complex piece of behavior can be understood as the concatenation and performance of each of its underlying production rules" (1993). Many of Anderson's experiments were of students learning introductory programming. So I took this to mean that learning to program is learning a bunch of simple steps. If learning programming is learning a bunch of simple steps, why are so many novices struggling and how can we help them learn these steps?

Clearly the importance of understanding how computers work is increasing in our society and is seemingly influencing every field and ultimately everyone. To me, this means there is increasing importance for everyone to learn introductory programming

to demystify and understand these machines we are building. Therefore this seems a worthwhile endeavor in which to spend a significant portion of my career. So, I began this work interested in applying my engineering skill and teaching skill to this problem of helping novices to learn to program while simultaneously spending time reviewing the research and developing scientific skill to determine what is really effective for teaching introductory programming.

General Problem

Multi-national, multi-institutional (MNMI) studies have shown that the challenges with novices learning to program is worldwide. McCracken et al's (2001) MNMI study summarized "…many students do not know how to program at the conclusion of their introductory courses." Lister et al's (2004) MNMI study showed "…many students lack knowledge and skills that are a precursor to problem-solving." They note that novices have "…a fragile ability to systematically analyze a short piece of code." McGettrick et al (2005) note "educators cite failure in introductory programming courses and/or disenchantment with programming as major factors underlying poor student retention" with dropout rates in computing disciplines as high as 30-50% at many institutions. McGettrick et al describe the Grand Challenge before computing educators: "Understand the programming process and programmer practice to deliver effective educational transfer of knowledge and skills."

Potential Solution

In 2010, Anthony Robins published his Learning Edge Momentum (LEM) theory (Robins, 2010) which gave an explanation regarding why many instructors seem to see evidence of those that can learn to program and those that cannot. In other words, an explanation for why we seem to see a bi-modal distribution of grades in our programming classes. Essentially, Robins argues that there is little or no evidence for those that can and those that cannot learn to program despite many years of looking for evidence. Robins contends that it is well known that we learn on the edge of what we already know. He argues that when learning programming, the concepts are unusually tightly connected when compared to other fields. If a student misses a concept then acquiring other concepts that depend on the missing concept becomes increasingly difficult creating negative learning momentum. Contrastingly, if a student is able to master a concept then the next concept is easier and so creates positive learning momentum. This learning momentum phenomenon may begin very early and over the course of a semester, Robins argues, this momentum results in instructors seeing a bi-modal distribution – some novices get it and some novices do not. An implication of Robins' Learning Edge Momentum theory is that by changing instruction to loosen the tight interconnection of concepts and reduce cognitive load we might be able to make introductory programming accessible to many more novices. Mayer and Moreno (2003)

describe cognitive science research regarding the challenge of cognitive load and suggest nine ways to reduce cognitive load with multimedia instruction.

Since the MNMI studies have more credibility than institution specific studies and take the focus away from issues specific to an institution, we wanted to be able to systematically test our efforts with a similarly diverse population. Due to the challenges with finding a significant number of diverse participants with some programming knowledge but not the knowledge we wanted to provide, we decided the best approach would be to find participants with no programming knowledge and teach them the first few concepts of introductory programming. To carry out our study we preferred the credibility of existing instruments rather than the time and resources involved with creating our own. However, Tew and Guzdial (2010) claim "The field of computing lacks valid and reliable assessment instruments for pedagogical or research purposes." Fortunately, there have been authors that have published their assessments and encouraged others to use them. Therefore, we decided to adapt and utilize Corney, Lister and Teague's (2011) questions for the very early, fundamental programming concepts of assignment and sequence that seemingly were difficult for novices in week 3 of a classroom course.

My Thesis

Application of cognitive load reducing methods to instruction increases novices'

ability to trace Java code that utilize the concepts of declaration, assignment and

sequence.

The rest of this dissertation is organized as follows. The Related Work chapter discusses

the difficulty novices have with the introductory concepts of assignment and sequence,

cognitive load reducing methods and other environments for learning programming.

The ReadJava Simulator chapter describes the adaptable tool we have built to assist

novices with learning to program. The third chapter, Experimental Results, describes

our experimental methods and results. Finally, the Conclusion summarizes the main

contributions and some lessons for the future.

2. Related Work

In this chapter we review work relating to the research reported here. First we describe the evidence that novices have difficulty with the fundamental concepts of assignment and sequence as this is important to show these are a relevant problem to focus our initial investigations. Second, we review methods for reducing cognitive load in instruction that we will be applying when preparing our instruction and that ultimately seem to have a surprisingly large effect on the results. Finally, as a part of our effort we have built a substantial tool, ReadJava simulator, to assist novices with learning to program. Many others have built tools and learning systems to assist novice programmers as well so we compare and contrast our tool with theirs.

New Programmers' Difficulty with Assignment and Sequence

We desire important and difficult introductory programming concepts that could feasibly be taught in a short amount of time to novices with no programming background in order to assess the effectiveness of our teaching methods with a substantial, diverse population of participants. Corney, Lister and Teague (2011) studied novice programmers in their first semester of programming and summarize "…the problems many students face with understanding code can begin very early, on relatively trivial code...these problems often go undetected until late in the semester."

They focused on the introductory concepts of assignment and sequence since code segments utilizing them seem trivial but, as they showed, are not for novices. They reported that 83 out of 227 students (36%) successfully answered all three screening questions in week 3 of their course. Of those that successfully answered the screening questions, about 47% successfully answered an explain-a-swap question and subsequently did very well on the end of semester exam. Those students that did not successfully answer an explain-a-swap question were not nearly as successful on the end of the semester exam.

Porter and Zingaro (2014) provide evidence that suggests that the relationship between early success with fundamental concepts and success in the course is due to those fundamental concepts being a part of most questions on a final exam. The fundamental concepts in introductory programming are assumed to be mastered and built upon quickly. If the fundamental concepts are not mastered early, then the difficulties of the students persist throughout the course.

We were interested in the impression of other computer science educators on the difficulty of the concepts of assignment and sequence. In February 2014, we selected the second screening question (adapted for Java with minor modifications) and asked members of the computer science education community what percentage of their students could answer the question correctly by week 3. The specific question in the

survey sent by email to the SIGCSE-members[1] mailing list with subject "Single Question

Survey" is shown in Figure 1.

Approximately what percentage of your students would be able to answer the following question correctly by week 3 in an introductory programming course at your institution?

```
Write the values in the variables after the following code has
been executed:
int a;
int b;

a = 3;
b = 5;

a = b;
b = a;
The value in a is: _____
The value in b is: _____
```

Figure 1: SIGCSE Single Question Survey question

There were 219 total responses of which 32 were blank. The rest of the responses

ranged from 0% to 100% as shown in Figure 2. Some of the members of the SIGCSE

community responded to the author directly, pointing out some of the variability in

ways courses are taught with comments such as "hadn't introduced assignment

statements by week 3", and "it is not possible to answer it for our introductory course,

which is taught in a functional language and does not cover assignment or mutable

variables".

---

[1] Special Interest Group on Computer Science Education (SIGCSE)

http://www.sigcse.org/membership/mailingLists
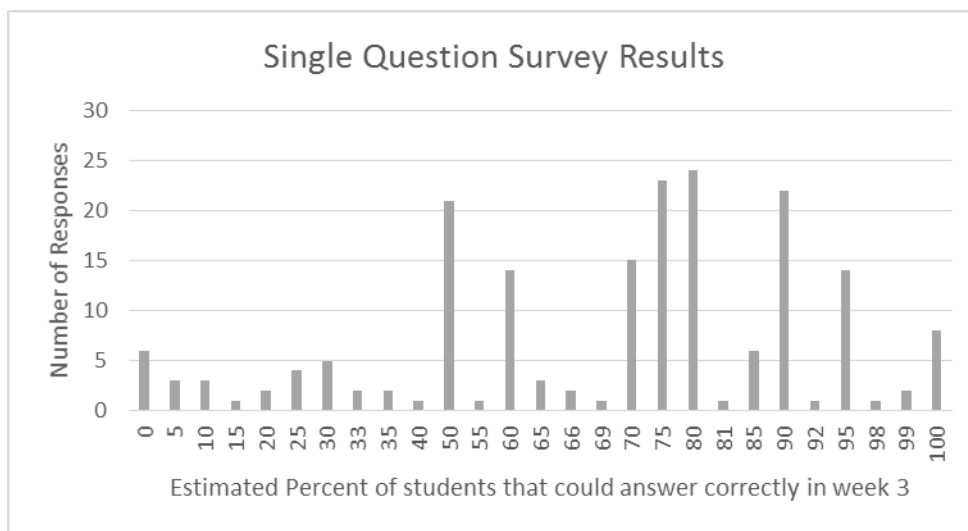
Figure 2: SIGCSE Single Question Survey results

The responses ranged from 0 to 100% reflecting the diversity of the students,

institutions, methods of teaching and the broad range of perceptions. The average of

the responses to the question in the Single Question Survey was 67% with a median of

75%. Interestingly, when Ahadi and Lister (2013) had asked students that same question

in week 3 and they found that 66% of students successfully answered it. In other words,

on average, specifically for the question we asked, many educators believe the concepts

of assignment and sequence are not trivial for their students.

Reducing Cognitive Load

An implication of Robin's Learning Edge Momentum theory (2010) is that if we can

design instruction that reduces cognitive load that may be a significant help to novices.

Mayer and Moreno (2003) describe meaningful learning as a deep understanding of the material such that the learner can apply the material to new situations measured using problem-solving transfer tests. Transfer tests ask the learner to solve new problems using the material. However, Mayer and Moreno state that "Meaningful learning requires that the learner engage in substantial cognitive processing during learning, but the learner's capacity for cognitive processing is severely limited"(p.43). According to Mayer and Moreno, a key challenge for instructional designers is "the potential for cognitive overload—in which the learner's intended cognitive processing exceeds the learner's available cognitive capacity." (p.43)

Mayer and Moreno describe three assumptions from cognitive science research about how the mind works: dual channel, limited capacity and active processing. Dual channels refer to our auditory and visual channels for processing verbal and pictorial information. Both the channels are very limited in terms of the amount of cognitive processing that can take place at one time. However, paying attention to new material, organizing it and integrating with current knowledge takes significant cognitive processing.

Mayer and Moreno (2003) summarize nine cognitive load-reducing methods for presenting materials (p.46):

Off-loading: Move some essential processing from visual channel to auditory channel.

Segmenting: Allow time between successive bite-size segments.

Pretraining: Provide pretraining in names and characteristics of components.

Weeding: Eliminate interesting but extraneous material to reduce processing of extraneous material.

Signaling: Provide cues for how to process the material to reduce processing of extraneous material.

Aligning: Place printed words near corresponding parts of graphics to reduce need for visual scanning.

Eliminating Redundancy: Avoid presenting identical streams of printed and spoken words.

Synchronizing: Present narration and corresponding animation simultaneously to minimize need to hold representations in memory.

Individualizing: Make sure learners possess skill at holding mental representations.

Our efforts investigate whether instruction designed specifically to minimize cognitive
load will be effective in helping students improve their understanding of programming
concepts. We applied cognitive load reducing strategies to our design of short
instructional materials to teach the concepts of declaration, assignment and sequence
to students that have never had programming training. Following the instruction, we
asked the participants to answer adaptations of the screening and explain-a-swap
questions of Corney, Lister and Teague (2011).

Environments for Learning Programming

Since at least the 1960's, there have been many efforts and tools to assist novices with
learning to program. Kelleher and Pausch (K&P) (2005) survey the programming
languages and environments since the 1960's and categorize them according to their
primary goal. By discussing how we would place our work within the K&P taxonomy, we
are effectively comparing and contrasting our work with all the systems within the
taxonomy. After we place our work within the taxonomy, we compare and contrast our
work with more recent learning systems. Figure 3 shows a partial taxonomy with the
gray nodes highlighting how we place the ReadJava simulator as a Teaching System,
focused on Mechanics of Programming, more specifically on Understanding Program
Execution and Tracking Program Execution.
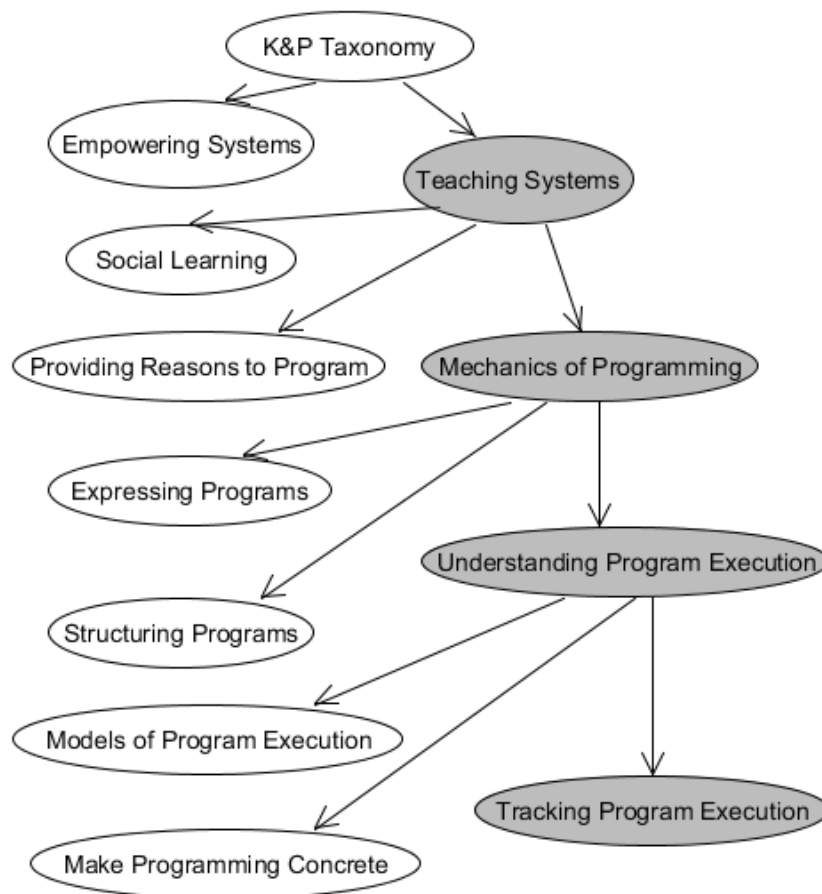
Figure 3: Partial K&P Taxonomy with gray categories showing how we place ReadJava.

Our work contrasts with the systems described in the K&P taxonomy in the following

ways. Our work is a teaching system rather than an empowering system that enables

learners to write code in a simplified environment such as Scratch (Resnick et al., 2009),

Alice (Kelleher & Pausch, 2007) and Greenfoot (Henriksen & Kölling, 2004). We agree

with Lister (2011) that while efforts such as these empowering systems may attract

students to programming and motivate them to begin to learn programming, they do not fully address the problem that learning to program is a challenging cognitive task. In contrast to some teaching systems within the K&P taxonomy, our system does not provide social support for learners to learn from each other or provide reasons to program. Our system is focused on assisting individuals with learning the mechanics of programming.

In contrast to other systems in the K&P taxonomy that also help novices with the mechanics of programming, our system is focused on helping novices learn to read, but not write code. Also our system focuses on specific instructions and code fragments and tracking program execution rather than organization and structure of code. Our system focuses on tracking program execution of specific Java instructions without the use of micro worlds or physical metaphors outside of a computer. Compared to other systems for tracking program execution, such as debuggers within professional development environments and The Teaching Machine 2 (Bruce-Lockhart & Norvell, 2007), our system initially provides an extremely simple model of a computer and memory that adds detail as the novice progresses.

In contrast to more recent systems such as Jeliot 3 (Moreno, Myller, Sutinen, & Ben-Ari, 2004) and UUhistle (Sorva & Sirkiä, 2010) that provide elaborate visual animations, our system simply highlights code and shows how memory contents change. Petre notes

that while visualizations are appealing, they are not necessarily helpful, are usually slower to transmit the same information and frequently require learning a secondary notation regarding layout, typographic cues and graphics. Relevance of the visualization, the structure and relationships are not obvious to everyone (Petre, 1995). Indeed, in one of our experiments we found that a substantial number of self-reported novices actually performed more poorly when given a visualization of the state of computer memory than when they just read the code and heard the narration.

Levy and Ben-Ari (2007) lament building visualization tools, stating "we work so hard and [teachers] don't use it". Naps et al (2003) surveyed SIGCSE[2] members on the top impediments for using visualization in their teaching. The top five impediments listed by response percentage are 1) 93%: time required to search for good examples, 2) 90%: time it takes to learn the new tools, 3) 90%: time it takes to develop visualizations 4) 83%: lack of effective development tools, and 5) 79%: time it takes to adapt visualizations to teaching approach and/or course content. In contrast to other tools, our system is intended to have no learning curve to use and no burden on the instructor to develop supporting materials. Since our method focuses so specifically on the meaning of specific Java instructions, the bite-sized chunks can be easily utilized as

---

[2] ACM Special Interest Group on Computer Science Education http://www.sigcse.org/

supporting materials by instructors utilizing any existing methods of teaching

introductory Java programming.

3.   The ReadJava Simulator

In this chapter we discuss why we chose to build our own tool, we describe the tool, the development approach and provide several detailed examples of use of the tool. For the tool description we include the architecture, design rationale, user interface and configuration. Our examples are chosen to illustrate how we have implemented important and difficult introductory computer programming concepts that we believe may be helpful to many novices. In the next chapter, we discuss experimental results using the tool with novices learning the first steps with programming.

Why Build Our Own Software?

Initially, we considered and began investigating adapting the NetBeans debugger to alter the memory view such that it would have minimal data appropriate for first step novices. However, we realized we did not want just a simpler view but desired a conceptual model which "…is not a mental model but an explanation of a system deliberately created by a system designer, a teacher, or someone else. Its purpose is to explain a system's structure and workings to potential users" (Sorva, 2012). The model we wanted did not have to be technically correct but should be appropriate for novices taking their first steps with learning to program.

To minimize the cognitive load of the instruction and loosen the tight binding between programming concepts, we wanted to be able to design instruction in bite-sized segments and relentlessly weed extraneous information. As noted by Bruce-Lockhart & Norvell (2007), when we program, the machine that we are giving instructions combines aspects of the computer, compiler and memory management. Initially for novices we want to hide the details of what each part of this system does. For example, we wanted the flexibility to simulate a program fragment rather than a complete Java program as well as reinforce the notion that programming is a mechanical process of executing one instruction after another. In Java, a strongly typed language, a variable must be declared before it can be used. However, the compiler handles the declaration and will allocate the local variables on the stack when the method is called. When executed step-by-step, the variables will be available as the method is called but before the body of the method is executed. In other words, to a novice the variables seem to appear before execution specifically reaches them. Explaining all this to a novice trying to write their first programs, we believe, helps lead to cognitive overload. Therefore, we felt the easiest path was to write our own simulator that would give us control of exactly how we wanted to present the Java instructions, even if a code fragment and even if not a technically correct implementation.

Architecture
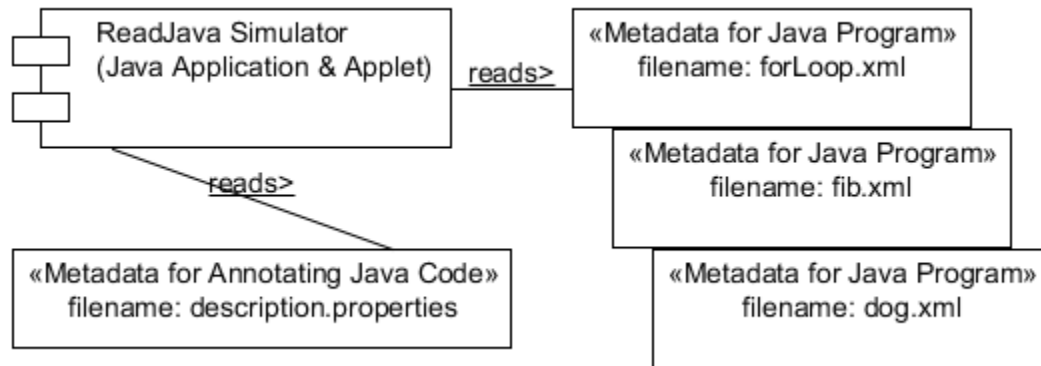
Figure 4: ReadJava simulator program and configuration files

The ReadJava simulator is implemented in Java and can be run as a desktop application

or applet[3]. On program startup, the simulator reads configuration information from two

files, an XML file containing the Java program and a text file that contains the

descriptions for the meaningful steps we have defined for Java programs, essentially

annotations for Java code. The Java program file, specified as a command-line or applet

---

[3] We were initially anticipating distributing the applet online to our participants with an audio clip rather

than creating the screencast. However, given the security concerns, warnings and configurations for Java

applets, participants would probably have had much higher frustration and much lower successful

participation rate. In addition, instead of mimicking the controls of a video to reduce cognitive load, we

were able to have the participants simply use YouTube video controls which they were likely already

familiar.

parameter, provides the Java program or program fragment to simulate along with various configuration options for the simulator. There is a Java program file for each example program which may contain more than one Java class. The metadata for annotating a Java code file is a list of name/value pairs (properties) that provides the textual descriptions for each step in the execution of a program within the simulator. Having the textual descriptions separate from the code makes them easier to modify as we investigate the best descriptions to provide to novices. Currently there is a single property file with annotations but we can imagine multiple files for different levels of novice programmers or perhaps with the text in different languages.

Figure 5 shows the ReadJava simulator we designed and built to explain and illustrate various introductory programming concepts for novices. Using Camtasia Studio software, we recorded screen casts with a narrator using the simulator that are shared with novices as bite-sized segments of instruction.
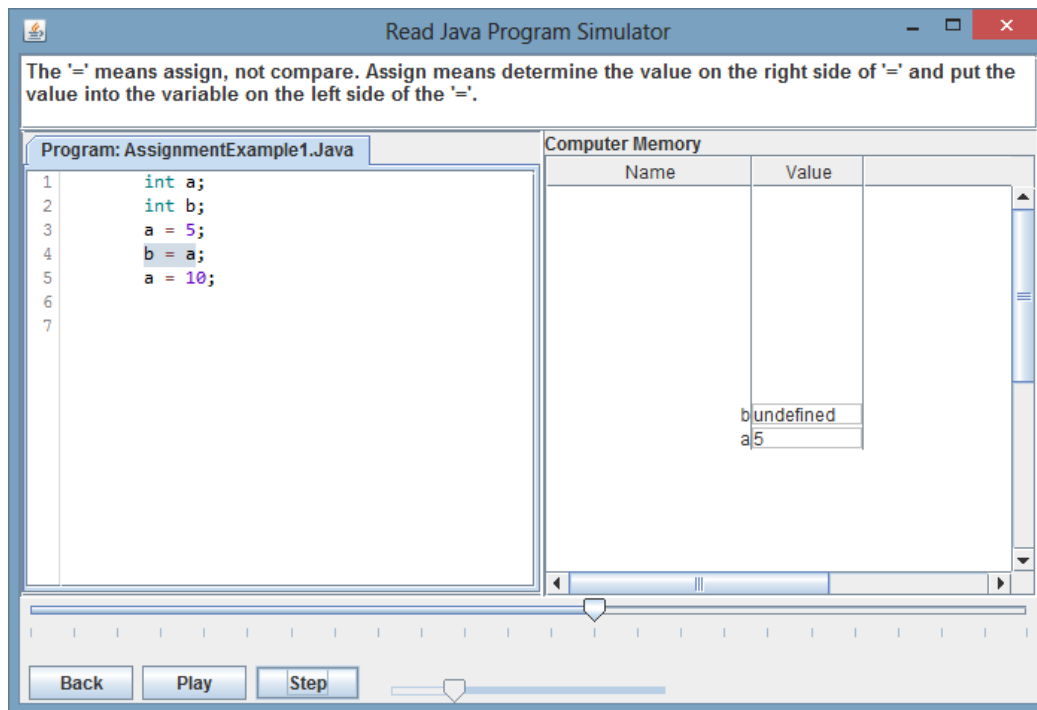
Figure 5: ReadJava Simulator executing a Java program fragment

Java Program Fragments

Even a simple Java program requires understanding many concepts. An example of the

simplest possible complete Java program that provides output and is typically the first

program described to novices is shown in Figure 6:

```
public class HelloWorld {
     public static void main( String [] args) {
          System.out.println("Hello World!");
     }
}
```
Figure 6: "Hello World" Java Program[4]

When executed, this "Hello World" program writes the characters `Hello World!` to an output device, such as a console window. Many of the keywords and symbols here such as `public`, `class`, `static` and `[]`, will not be covered in any depth until later in a first semester programming course. In order to minimize the potential for cognitive overload, our solution allows program fragments (incomplete Java programs) to be executed, such as the simple example shown in Figure 7 containing the concepts of declaration, assignment and sequence.

```
int i;
i = 2;
```
Figure 7: Java Program Fragment

Metadata for Java Programs

When the ReadJava Application runs, an XML file containing the code to run is specified on the command line (or as an applet parameter when run as an applet). An example XML file is shown in Figure 8

---

[4] https://en.wikipedia.org/wiki/Hello_world_program   retrieved May 6, 2013.

```
<topic>
    <filename>AssignmentExample1.Java</filename>
    <mainClass></mainClass>
    <program><![CDATA[    int a;
    int b;

    a = 5;
    b = 10;

    a = b;
    b = 7;
        ]]></program>
    <showMemory>true</showMemory>
    <memoryColumns>Name,Value</memoryColumns>
    <showOutput>false</showOutput>

<showLiteralAtFirstReference>true</showLiteralAtFirstReference>
    <showVarAtDeclarationTime>true</showVarAtDeclarationTime>
</topic>
```
Figure 8: XML configuration file for simulator

The empty tag `<mainClass>` indicates that this is a Java code fragment and not a

complete Java program with a main class and main method. In the case of a fragment,

the name "(default)" is used within descriptions that specify the class and method

names.

The value for the tag `<showMemory>` indicates whether to show the computer

memory view or not. For some concepts or potentially some learners, the memory view

may not be helpful. Mayer and Moreno (2003) note learners with high spatial ability

may benefit more from simultaneous presentation of visual and auditory information

than low spatial ability learners.

The tag `<memoryColumns>` describes the columns to show in the computer memory view, in this case the Name and Value columns. We also have a Description column that contains terms that describe the various memory areas. Weplan to add Address, Hexadecimal Value and Binary Value columns as they may be useful for more advanced novices to learn more detail about how values are stored in computer memory. The `<showOutput>` tag determines whether to show the Program Output view.

The `<showLiteralAtFirstReference>` tag indicates whether to delay showing the literal in memory until execution has reached the literal. Since literals are processed by the compiler they will be loaded into memory when the Java class is loaded into the Java Virtual Machine (JVM). To a novice that may have heard of but has not yet mentally organized or integrated these concepts, having the literal just show up in memory prior to execution may reinforce the notion of "a hidden mind somewhere in the programming language that has intelligent interpretive powers" (Pea, 1986). For example the author, when teaching novices in a classroom, has had novices express confusion related to why one can type a "1" in a program and it just goes into memory, while when the program reads a "1" from a file it must be specifically converted to an "int" to be stored in memory. If we show the literal in memory at the time execution reaches the literal then we are showing a naïve view but have the opportunity to provide an explanation for the conversion of the literal from typed characters to the

binary format in memory, which otherwise might just seem to occur by the "hidden mind". Whether or not we show literals loaded into memory when the class is loaded or at runtime when the literal is reached as a part of execution, the resulting execution is the same.

Discussion of literals was included in the videos in Learning Survey 1 but was not included in the videos in Learning Survey 2 as it was determined to be an unnecessary detail for the first few minutes of instruction to teach novices to trace code. Removing this detail from our Learning Survey 2 videos is an example of applying the weeding cognitive load reducing method.

The `<showVarAtDeclarationTime>` tag, when true, means the simulator should wait until the declaration statement for a variable before showing the variable in the computer memory view. Again, this is a naïve interpretation intending to reduce the cognitive load by explaining certain concepts in a simple way initially to create a bite-sized instruction segment.

Metadata for Annotating Java Code

The textual descriptions that are shown to the user at each step of Java code execution are stored in a text file with a single line per item. Each item has a name (property) and value separated by '=' sign. The value is a string that contains parameters that are

bound to values at runtime. An example of a parameter is $varName$ which will

contain the specific variable name at runtime. A partial example of this file is shown in

Figure 9.

```
37
38   literalString=Allocating memory for an instance (object) of the java.lang.String class for '$value$'.
39
40   integerLiteral=Integer literal with value $value$.
41
42   elementValuePair=Saving the value '$value$' into the variable '$varName$'.
43
44   statementIf.conditionTrue=Since the if conditional expression is true, executing the statement within the if statement.
45
46   statementIf.conditionFalse.withElse=Since the if conditional expression is false, executing the statement within the else part
```

Figure 9: Partial view of text description properties file.

A special property 'config.showParams' normally has a 'false' value. This particular

property is intended to be helpful for instructors that would like to make changes to the

text shown for specific steps. Since there may be similar text for different steps in

execution, knowing which text to change within the file may not be obvious. By setting

the 'config.showParams' property to "true", the property name will be shown along

with the text to the user of the simulator. An instructor can simply search the properties

file for the unique property name in order to find the text the instructor wishes to

change.

Controls

To reduce or eliminate the need for learners to learn how to use the tool, we designed it

to be similar to other tools that the learners probably already use. For example, playing

a video seems similar to our task of watching a simulation of a Java program being executed that visually illustrates one instruction at a time. They both have a large visual component, a starting point, and an ending point. Both could be paused, rewound and started again. A video can be stopped at any time, and restarted at any point, forward or backward, whenever a user desires. Therefore, we have designed the controls for our simulator to be similar to video controls. For example, there is a Play button in the simulator that behaves the same way as a play button for a video. Additionally, there is a timeline showing progress of the execution, similar to a timeline on a video. Clicking on the timeline at any point changes the execution of the program to that point, either forward or backward.

Consistent with Webber (1996) we avoid teaching input and output initially to keep the learner focused on learning computation concepts. Avoiding input also allows us to execute the program completely and therefore allows the user to click forward on the timeline. The steps are shown to the user as the user wishes by either jumping forward or backward. Some differences between our simulator and a video are that it may be helpful to explicitly advance through each step of the simulation at the pace of the user rather than a preset time. Therefore, additional buttons are provided labelled Step and Back for that purpose.

We initially intended to have participants utilize the Java applet directly. However, due to applet security concerns, configuration and the variety of browsers our participants would have, this did not seem to be feasible. Therefore, we decided to record use of the simulator with a software product, Camtasia Studio, which includes recording audio. Videos hosted on YouTube are widely supported by browsers and therefore we could avoid many issues. As we record a screencast of the simulator with a narrator, the novice is actually just using video controls and therefore should have little to no learning curve for use of the tool.

Examples

ReadJava is able to simulate some of the most important and difficult concepts in introductory programming. Our Looping example can be utilized to describe how to trace control flow and help with understanding loop variable scope. Our Method Call and Recursion example can be utilized for illustrating parameter scope, procedure design and tracing and designing recursion. Our Class Instantiation and Class Variables example can be utilized to explain classes and objects, static fields and methods, polymorphism, inheritance, memory model, references and pointers.

Looping

Figure 10 includes an example with loop code. Tracing control flow and understanding

loop variable scope are in the top 32 important and difficult concepts identified by

Goldman et al (2008). Prior to the loop example shown in Figure 10, we are assuming

the learner has had bite-sized learning of literals, including String literals, the "+"

concatenation operator, writing out output (e.g., System.out.println), comparison

operators (e.g., <=) and the increment operator (e.g., ++). In Figure 10, the Program

Output view is shown along with the Description column in the computer memory view.

The Description column names various regions of memory and provides terms that

describe the various memory areas. For example, regions of memory are denoted

"Literals", "Instances", "Temporary Storage" and "Method Variables (Stack)". The

literals are shown as the program starts up, rather than waiting until the literal is

reached during execution. The Instances area includes the String literals which are

instances of the String class. Temporary Storage is way of referring to temporary areas

of memory without reference to registers, machine code, or memory management.

Method Variables, shown at the bottom of the Computer Memory view, are essentially

the program stack showing the local variables and parameters for methods. In addition,

this area also shows the return values from previous method calls within the current

method. In the Method Call and Recursion example, we will cover this region in more
detail.

Note that an instructor can hide the Literals and Instances, and even the Description
column, via the configuration file in order to focus attention on the way a for loop works
in Java. In the bottom right corner of Figure 10 are the Edit Program and Compile & Run
buttons. Our focus is on reading Java and not writing and so this capability is not
intended for first step novices. However, we may offer this capability to more advanced
learners.
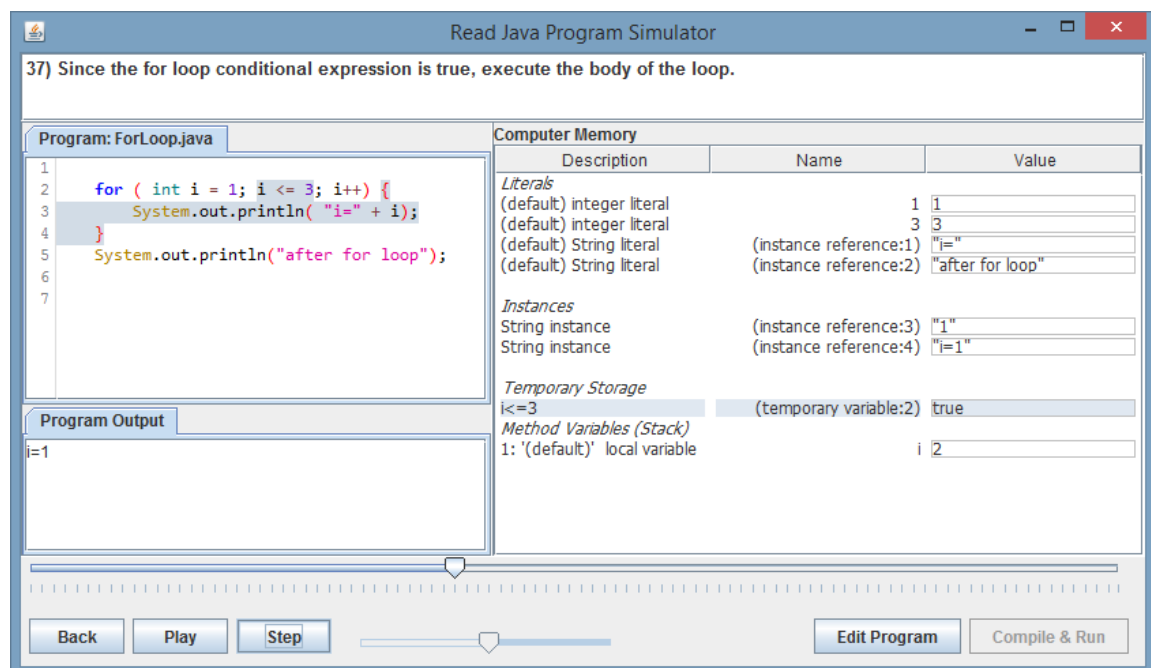
*Figure 10: Loop code fragment example, showing output, local variables, instances and
literals.*

Method Call and Recursion

Figure 11 shows an example of a recursive method call. Understanding parameter

scope, procedure design, and tracing and designing recursion are all in the top 11 most

important and difficult programming topics (Goldman et al., 2008). In Figure 11 is a

Fibonacci class with the fib static method. With this example, the novice can see

method calls and recursion in action. As a novice can see, the Method Calls area of

memory grows taller with each call and shrinks with each return. The return values

from methods called within the current method are shown in memory. In the

Description column, the numbers preceding each memory item indicate with which

method call they are associated. The value of the parameter at each point in time is also

shown. A narrator could step through bite-sized aspects of this example to highlight

various aspects such as how the stack grows and shrinks. A narrator can point out that

the recursive calls seem to be making the same method calls to calculate the same

values, over and over. We anticipate that this kind of explanation demonstrating

recursion may be helpful for novices to see the effects of recursion and also able to

realize when iteration may be more efficient. We also anticipate that having a

contrasting demonstration with an iterative version of the code for calculating the same

Fibonacci sequence may be helpful for a novice.
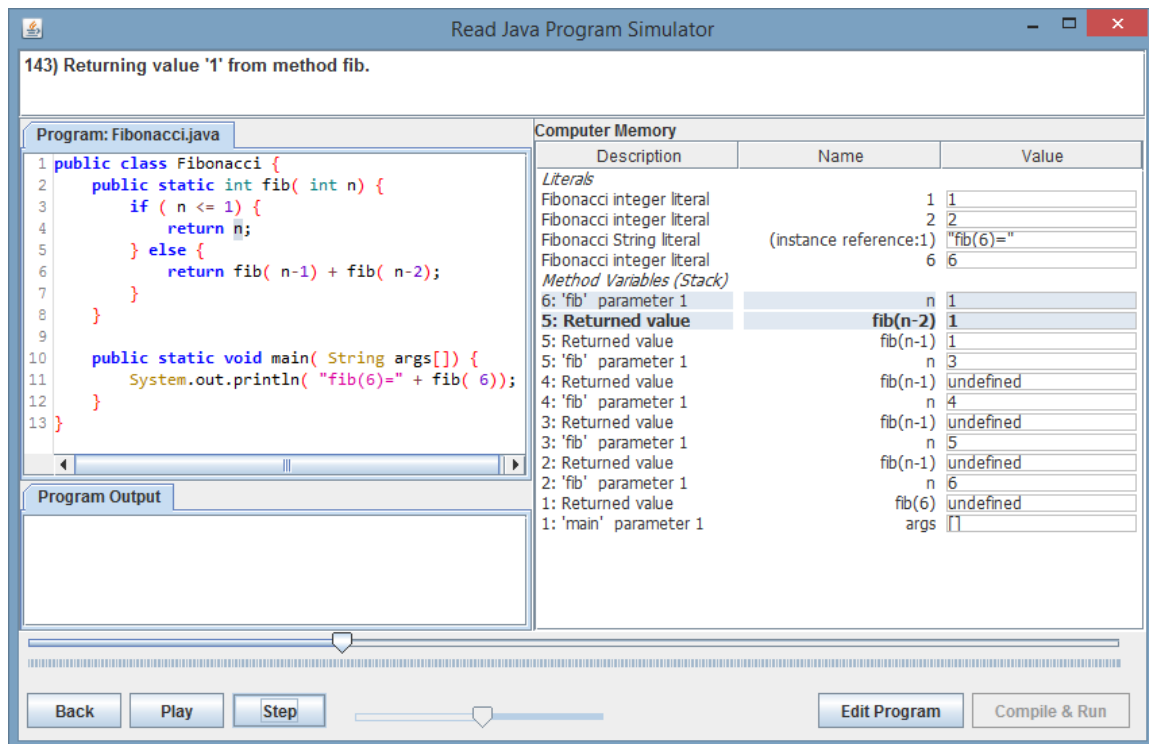
Figure 11: Complete recursive program showing local variables on stack and a value being returned from a previous recursive call.

Class Instantiation and Class Variables

Figure 12 shows a class instantiation example. The difference between classes and objects, static fields and methods, and polymorphism are in the top 32 most important and difficult programming topics while inheritance, memory model, references, and pointers are in the top 11 (Goldman et al., 2008). The example in Figure 12 can be utilized to discuss these concepts. Of course, it will likely take a number of bite-sized

guided video examples to illustrate these concepts such that novices are not overwhelmed.

In Figure 12 there is a Dog class that inherits from an Animal class. There is a third class, TestDog, with a main method which declares and allocates an instance of a Dog. In this example, the class variable numDogs is shown when the Dog class is loaded into the simulator to emphasize that it exists even without an instance of Dog being created. As a "new Dog()" is initialized each initializer and constructor is highlighted and executed in turn showing the various instance fields being initialized. This highlighting should help a novice to understand the difference between class (static) and instance (non-static) variables as well as realize that an instance of class Dog inherits the instance variables of the parent class Animal. At the moment in time the simulation is shown in Figure 12, the new instance of Dog has been allocated and initialized but control is still within the Dog constructor. In the next few steps the control will return to the 'main' method and the reference to the instance "(instance reference:9)" will be saved to the variable 'd'.

Figure 12: Class instantiation example showing class variable and instance fields in the process of be allocated and initialized.

Discussion of Design Decisions

Our primary goal is to reduce cognitive load for novices learning to program. A secondary, but essential goal, is to create a learning tool that instructors and learners will want to use. Here is our rationale for some important design decisions.

Java Programming Language

First, why use Java specifically in our instruction? Pears et al (2007) reviewed studies that describe factors that affect language choice in computer science education and found that external factors such as market appeal, industry demand and student demand are some of the most important factors in choosing a language. The Tiobe index[5] ranks the most popular programming languages using search engines and is updated monthly. Over the last 10 years Java has been one of the top three most popular languages. Currently, Java slightly lags C to be the second most popular but seems to be stable at that ranking with almost double the popularity of the next highest ranked languages C++ and Objective-C. We anticipate that Java will continue to be used widely in both industry and within Computer Science departments and so we chose to utilize it.

---

[5] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html retrieved May 4, 2013

Reading Code vs. Writing Code

Venables, Tan and Lister (2009) found that the combination of tracing and explaining skills leads to skill in writing code. So, to reduce cognitive load we do not focus on solving a problem utilizing Java but focus on learning to read and trace Java instructions (segmenting and weeding in load reducing terms). Our instruction must show the Java code, but does not need to allow novices to edit the code or handle the incorrect code a novice might produce. Intuitively, if we want to teach someone to write in a language, such as English for example, it would be helpful if they knew some English words and had seen a number of examples of how they are combined to form a meaning prior to asking them to write English. They would not need to know a lot of English but have a working knowledge of some words and how to put them together.

Explanation and Highlighting of Each Instruction

The ReadJava simulator highlights each Java instruction as it is being executed along with showing the text explanation of a step and the memory contents at the point the step is executed. Highlighting and giving the explanation is applying the signaling and segmenting cognitive load reducing methods. Highlighting is signaling to focus the attention of the learner. Giving a specific explanation for a step is a form of segmenting as this is providing a very small unit of instruction at a pace controllable by the learner.

Bruce-Lockhart & Norvell (2007) note that when we program, the machine that we are

giving instructions combines aspects of the computer, compiler and memory

management. A literal or naïve interpretation of the instructions should be made that

would be most appropriate for a novice learning the meaning of programming

instructions for the first time. Essentially we are suggesting reducing the tight coupling

between concepts. For example, compilers frequently perform optimizations that may

influence the actual order of execution. One optimization is to allocate the memory for

all the local variables for a method at the time the method is called. However, if our

simulation shows that the local variables are allocated prior to the execution step

reaching the instruction that declares the variable, this could be quite confusing to a

novice about what Java instruction is actually responsible for the allocation of the

variable. Therefore, we start with the simple interpretation that a variable is allocated

when the declaration is reached during execution. After a student advances in

understanding, then additional detail and complexities can be explained. A literal

interpretation of the code now with more detail later are examples of the segmenting

and weeding load reducing methods. That there is a tool called a compiler that has

converted the code to another form that is executed by a not real but virtual machine is

too much extraneous detail for a novice that has the pre-existing assumptions

mentioned by Pea. These details can be added, in a cognitive load appropriate way, after the literal interpretation is learned.

Learning to trace control flow through program execution is one of the top 32 most important and difficult topics (Goldman et al., 2008). As du Boulay (1989) notes "Loops cause beginners all kinds of trouble". These troubles include that the loop control variable is incremented each iteration of the loop, that the conditional expression involving the loop control variable will change each time through the loop, and that the loop does not terminate at the very instant the control condition changes but will terminate the next time the control condition is checked. Thus our solution highlights each instruction as it is executed to focus the learner's attention as well as demonstrate control flow during execution.

As Mayer (1997) notes for multimedia learning, a graphic should be coordinated with the text explaining the graphic. The graphics, in our case, are the highlighted Java code. Our solution provides a textual explanation that describes the meaning of the highlighted code, suitable for a novice just beginning to learn to program.

Optional Model of Computer Memory

A model of computer memory seems very important and how we model computer memory seems as important. Our analysis, in Appendix A, of Goldman et al's (2008)

important and difficult programming fundamentals topics suggests about half seem

directly related to understanding how Java instructions influence the contents of

computer memory, thus effectively presenting the contents of computer memory for

Java instructions might be very helpful for novices. Ben-Ari (1998) asserts "students do

not have an effective model of a computer" and "models must be explicitly taught". In

his study of how novices learn computer programming, Mayer (1981) writes "a concrete

model can have a strong effect on the encoding and use of new technical information by

novices". We are intending the computer memory to be a conceptual model which "is

not a mental model but an explanation of a system deliberately created by a system

designer, a teacher, or someone else. Its purpose is to explain a system's structure and

workings to potential users" (Sorva, 2012). How we design the computer memory

model seems critical as Petre (1995) notes that visualizations are appealing however

they are not necessarily helpful, usually slower to acquire the same information and

frequently require learning a secondary notation regarding layout, typographic cues and

graphics. Petre allows that novices "might benefit from a more constrained system in

which secondary notation is minimized, in order to reduce the richness and the

potential for mis-cueing and misunderstanding." Thus, consistent with cognitive load

reducing methods, we start with a simple model of computer memory that can be

enhanced as a novice gains knowledge and understanding. The simplest concept of

memory to us is that a memory location has a name and a value. A list seems an

appropriate way to show several name/value pairs. We anticipate being able to add

additional fields, such as addresses and binary values, to the memory locations as a

novice learns. Thus a list of name/value pairs seems a good starting point for a memory

model and is straightforward to implement. Allowing the instructor or the novices to

adapt the simulator by showing or hiding various aspects is important. For example,

having the computer memory model visible is optional as it shows significant

information and may be a distraction. In one experiment, to be described here, some

self-reported novices performed more poorly when given a visualization of the state of

computer memory than when they just read the code and heard the narration.
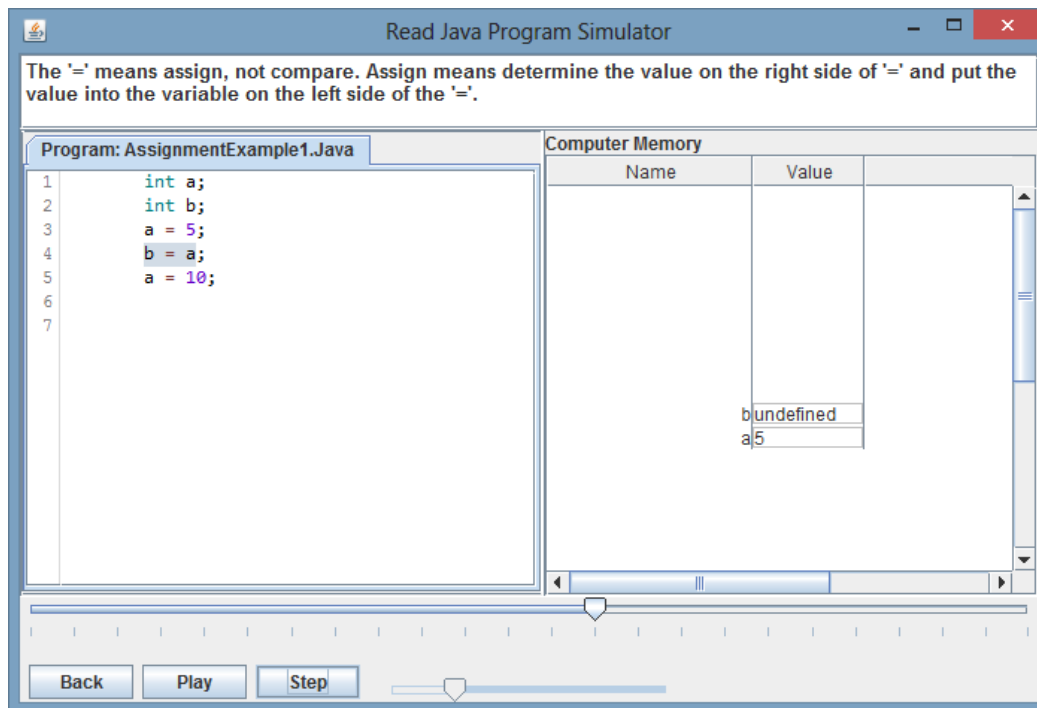
Figure 13: ReadJava Program Simulator executing a Java program fragment

Use of Multimedia

Clark and Mayer (2011) note that switching attention between a visual and the text that explains the visual, such as in a textbook, can overload human cognitive abilities particularly when the concepts are new or complex. When a visual is the focus of words then the words should be spoken simultaneously with the visual and the words should not be in written form. "The psychological advantage of using audio presentation is a result of the incoming information being split across two separate cognitive channels—

words in the auditory channel and pictures in the visual channel – rather than concentrating both words and pictures in the visual channel."

Therefore, we utilized Camtasia Studio, a software application that records the computer screen and audio from the computer microphone, to create a video. The video just shows the Java instructions and the computer memory model portions of the simulator with a narrator describing the meaning of the instruction thereby offloading text from the visual channel and adding narration to the auditory channel. In other words, we are applying the offloading and synchronizing cognitive load reducing methods. Since the learner does not need to learn the controls of the simulator but will use the likely more familiar controls of a video player, we have reduced the cognitive load further.

As noted by Mayer (2004), guided methods of learning are more effective than pure discovery learning where a learner is free in interact within a learning environment without guidance. Having these recorded, guided explanations of example programs executing in the simulator is likely more effective than expecting the learner to discover important concepts by just exploring the examples with the simulator on their own. While we may eventually develop the simulator as a product to be shared, we are not convinced that should be a primary focus of our work. Levy and Ben-Ari (2007) lament building visualization tools, stating "we work so hard and [teachers] don't use it". Naps

et al (2003) surveyed SIGCSE[6] members on the top impediments for using visualization in their teaching. The top five impediments listed by response percentage are 1) 93%: time required to search for good examples, 2) 90%: time it takes to learn the new tools, 3) 90%: time it takes to develop visualizations 4) 83%: lack of effective development tools, and 5) 79%: time it takes to adapt visualizations to teaching approach and/or course content. Making the simulator available, even if proven effective in some cases, will not necessarily be beneficial to many. However, guided instruction as we have described above with essentially no learning curve for either instructors or students, other than the content itself, may be much more beneficial to many.

---

[6] ACM Special Interest Group on Computer Science Education http://www.sigcse.org/
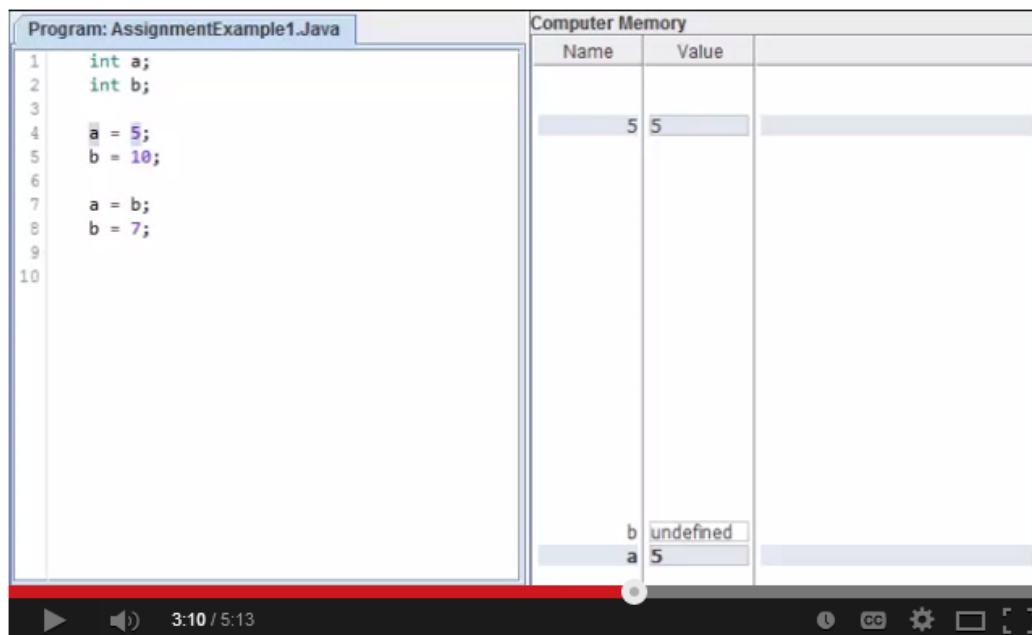
Figure 14: YouTube video of ReadJava simulator in use

Development Approach

We used an iterative, depth-first approach to building a complex tool that would allow

one to test various hypotheses for improving computer science education as well as

deliver computer science instruction. From our teaching experience and published

important and difficult introductory programming topics (Goldman et al., 2008) we

defined a set of example programs that our simulator should be capable of handling.

These include a short program fragment with declaration and assignment statements, a

looping program fragment, a complete recursive method call program and a complete,

multiple class object instantiation program. We implemented the features in the compiler and interpreter necessary to simulate these programs.

Our design is intended for straightforward extension. For example, we utilized a complete Java 1.6 grammar and simply stubbed the language elements in the compiler and interpreter not necessary for our examples. We implemented String concatenation ("+") as it was used in the looping and method call examples. We implemented addition ("+") and subtraction ("-") as they were used in the recursive example. We stubbed multiplication and division as they are not utilized. We implemented method calls and return values as they are necessary for our recursive method implementation. We implemented constructors to illustrate the multiple constructor calls necessary to create an instance from a class that inherits from other classes as illustrating this process is important and difficult for novices. Our implementation of polymorphism simply compares the method name, number and exact data types of each parameter to determine the method to call. We have not implemented data type coercion. Our specific examples did not use double, float, short or byte and so those are just stubbed where appropriate in the code. There is very limited error handling of incorrect Java example code as we assume correct Java code examples and do not intend to support novices' development of code in the simulator in the near future.

This iterative, depth-first approach has enabled us to begin experimenting, validating

and adjusting our approach and tool with merely substantial development hours for a

single software engineer, the author. A complete Java simulator that works for any Java

program would take many development years with many developers. Over time and

with others help this may happen, but more iterative research is necessary to prove that

this size of effort is worth the resources for this particular project.

Implementation

We used a parser generator, ANTLR[7] (Another Tool for Language Recognition), to build a

parser for Java 1.6 based on a Java grammar available from www.ANTLR.org. We

inserted additional grammar rules into the Java 1.6 grammar to allow for Java program

fragments to be parsed that would not normally be allowed, such as the examples that

just include Java instructions illustrating declaration, assignment and sequence. For Java

fragments, the interpreter internally creates a default class and default method that it

utilizes to hold the statements in the fragment.

---

[7] Terence Parr designed and built ANTLR and his books The Definitive ANTLR 4 Reference (Parr, 2012) and

Language Implementation Patterns (Parr, 2011) have been instrumental.

Utilizing the ANTLR parser, we implemented a compiler with two phases: a definition phase and a reference phase. The definition phase builds a symbol table from the classes, fields, parameters and variables declared in the Java instructions. The reference phase resolves the references to the symbols in the symbol table. All information from compilation is stored in internal objects.

Following compilation, the simulator executes an interpreter that begins executing the Java instructions utilizing the parse tree and symbol table created by the compiler. Meaningful steps in the execution are recorded as a list of steps. Meaningful steps are those that we choose to highlight in the code, illustrate in computer memory and describe with text. Deciding what is meaningful is an ongoing process. Each meaningful step recorded is considered one increment of time. Every change to a variable value during execution has the current time increment recorded with it. Note that at this stage, we do not provide a means for users to input information into example programs. Therefore, they can be run to completion after they are compiled.

The novice is able to view the execution both forward and backward by navigating this list of steps using the timeline. Since each particular step corresponds to a particular time, the values of the variables at that time can easily be retrieved and displayed.

4.   Experimental Results

We prepared and carried out three experiments. The first experiment, Learning Survey 1, was designed to directly compare the instruction with a memory model, to instruction without a memory model. We hypothesized that the memory model would have some positive effect. The results of Learning Survey 1 indicated that there may be a small positive effect in some cases but was not statistically significant for our sample size. We anticipated that by improving the instruction applying the cognitive load reducing methods further we would possibly see improved results for the memory model.

In the second experiment, Learning Survey 2, we directly compared the improved instruction with the memory model and without the memory model along with the ability of participants to answer the questions without any instruction. The results of Learning Survey 2, indicated that instruction was effective but that use of the computer memory model seemed to have a small, detrimental effect in some cases. We noticed when comparing the results of the Learning Survey 1 and Learning Survey 2 that there was a substantial improvement from the first survey to the second seemingly due to the improved instruction. While we anticipated that reducing the cognitive load would be helpful to explaining the concepts, particularly to participants that had no expressed

interest in learning to program, we did not anticipate such a large difference between the two sets of instruction.

Since the first and second learning surveys were carried out about four months apart, we had advertised differently for participants and had paid different amounts for participation, the comparison of results between the first and second experiments could be suspect. So, we carried out a third experiment, Learning Survey 3, which directly compared the instructional without memory model videos from the first two experiments. The results of Learning Survey 3 indicated that the shorter, improved instruction from the second experiment was significantly more effective that the instruction from the first experiment.

Each experiment was a learning survey which consisted of participants watching one of a set of videos and then answering some questions that required applying the content of the video. The groups that watched each video were randomly chosen by the survey software and all the questions on a particular survey were identical for all participants.

Method

Learning Materials

We designed and built an adaptable Java program simulator, called ReadJava, applying the cognitive load reducing methods to assist novices with learning introductory computer programming concepts.

ReadJava simulates Java program fragments to allow us to create bite-sized code instruction segments and eliminate extraneous material (segmenting and weeding). ReadJava provides simple Next and Back buttons to step through the code fragment an expression at a time. Each expression is highlighted as it is executed (signaling). A teacher or a student can choose to see optional information such as an explanation of the step and whether to show and the detail to show within a simplified computer memory representation (signaling, weeding, aligning). See Figure 15.
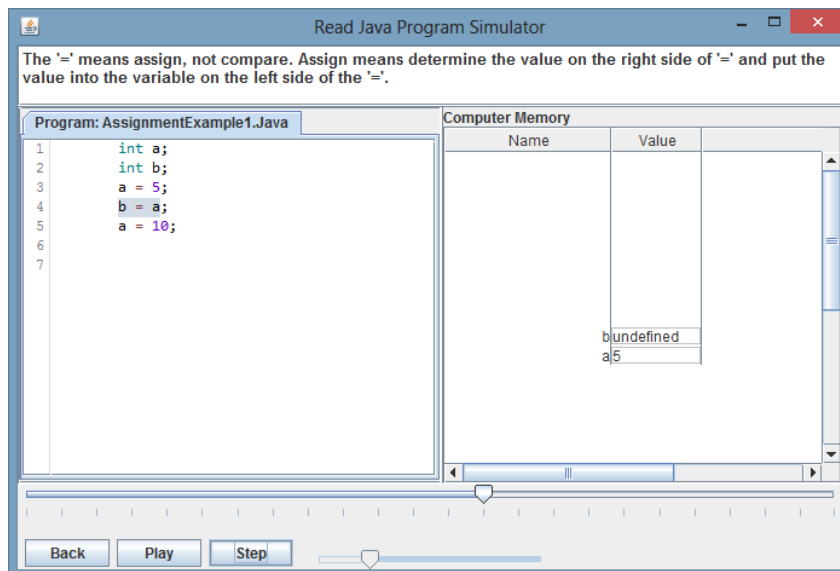
Figure 15: ReadJava Program Simulator executing a Java program fragment

Following the cognitive load reducing principles, we simplify the presentation further, for our specific example here, by creating a screen cast just showing the steps being highlighted and have a narrator explain the meaning of each step (offloading, eliminating redundancy, synchronizing). See Figure 16. Making a video also simplifies the use of the tool for the learner by making the controls something they are probably already familiar with (we use YouTube[8] to host the videos) and therefore eliminates the need for the learner to learn to control a new tool (weeding). Within the video, we use some notations to cue the learner (signaling).

---

[8] http://www.youtube.com/
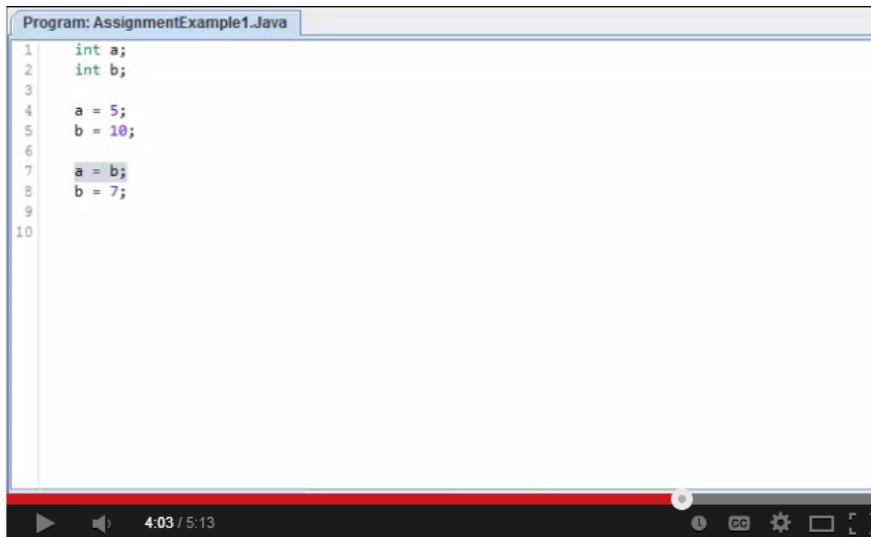
Figure 16: Screen cast of ReadJava simulator that participant views.

Learning Surveys

We utilized Qualtrics Survey software to create online surveys that included

demographic, prior knowledge, a catch trial, tracing and relational response questions.

Demographic questions were age, gender, education and location. We used the

question "Have you ever had training on how to write a computer program either in a

classroom or online?" to separate out those participants with no prior programming

training. To assess whether the participants were carefully reading the questions we

asked the catch trial question "Have you ever had a fatal heart attack while working at

your computer?" and eliminated participants that did not answer "no" (Paolacci,

Chandler, & Ipeirotis, 2010).

The survey software has the capability to randomly choose one of a set of videos to show a particular participant. In the first learning survey, we had two conditions: 1) a video with narrated instruction that showed a computer memory model 2) a video with identical narration but did not show a computer memory model.

We analyzed the results of the first learning survey, applied the cognitive load reducing methods relentlessly and then created a second learning survey with three conditions. The conditions were 1) a video with narrated instruction that showed a computer memory model, 2) an identical video with narration except without the computer memory model, 3) a short video with brief narration but without any content instruction. The survey is the same in all three conditions.

There was a substantial improvement in performance on the tracing questions between the first survey and the second survey for the participants in both conditions that had narrated instruction without the computer memory model. To directly compare the two instruction-without-memory-model videos from the first two surveys we prepared a third learning survey.

Our tracing and relational response questions were adapted from the screening and explain-a-swap questions, respectively, described by Corney et al (Corney et al., 2011; Teague, Corney, Ahadi, & Lister, 2012) from their week 3 test that they shared and

encouraged others to utilize. We consider these transfer tests since they require the

learner to apply the material to new problems and not just recall the material. Their

goals for the first three screening questions were to assess whether students

understood variables and assignment, could trace code of similar complexity and

approached the test seriously. They identified code that swaps values between two

variables as the simplest, non-iterative code that tests relational reasoning. They ask

students to explain-in-plain-English the purpose of three lines of swap code in their

questions 4 and 5. Their question 4 has an explanation and an example of the kind of

response they are expecting.

We are interested in utilizing the Java language so we adapted the questions to Java as

they had done in the replication paper (Teague et al., 2012). Since these questions were

our focus we referred to them as tracing questions rather than screening questions. As

Java is a strongly typed language, we felt it was important to declare variables before

they were utilized therefore, we added declaration statements.

In the boxes provided below, write the values in the variables after the following code has been executed:
 	
``
r = 2
s = 4
r = s
``
The value in r is ⬚ and the value in s is ⬚
	In the boxes provided below, write the values in the variables after the following code has been executed:
	
	``
	int r;
	int s;
	
	r = 2;
	s = 4;
	r = s;
	``
	The value in r is: ⬚
	and the value in s is: ⬚

Screening Question 1 (Corney et al., 2011)	Our Tracing Question 1 (see appendix for actual survey questions)

Table 1: Comparison of screening question 1 in Python to our tracing question 1 in Java

Participants

A benefit of multi-national, multi-institutional studies (Lister, Adams, Fitzgerald, Fone, Hamer, Lindholm, McCartney, et al., 2004; McCracken et al., 2001) is a focus on general introductory programming education rather than education at any particular institution which may have characteristics such as location or admissions policy that attracts some students and faculty over others. As an individual researcher at a single institution we desired to find a way to have results independent of a particular university. Therefore, participants were recruited on Amazon Mechanical Turk[9] (AMT), an online labor market.

---

[9] https://www.mturk.com/mturk/. The name Mechanical Turk comes from a chess-playing "machine" from the 18th century. This machine actually utilized a hidden chess master. Amazon's Mechanical Turk

On AMT, requesters post jobs, called human intelligence tasks (HIT), and workers select HITs to complete for pay. In our case, we, the researchers, are the requesters and the workers are our participants. The task is watching the video and answering the questions in our survey.

Learning Survey 1

In the first learning survey we have two conditions: instruction with a computer memory model and instruction without computer memory model.

Learning Materials

For our first learning survey, we created two 5 minute 13 second videos applying the cognitive load reducing methods to explain a code segment similar in complexity to our three tracing questions. First, for the instruction with computer memory model video[10], we used Camtasia Studio[11] to record the author using the ReadJava simulator showing

---

web service is designed to hide human workers that do tasks for which machines are not suited. (see http://en.wikipedia.org/wiki/Amazon_Mechanical_Turk)

[10] Survey 1: instruction with memory shown: https://www.youtube.com/watch?v=ObZ7mA5EkuI

[11] Available from www.TechSmith.com

the computer memory model and narrating each step of the execution. To create the instruction without computer memory model video[12], we copied the original and visually removed the computer memory model using the Camtasia Studio video editing capability. This procedure results in videos that have identical length and identical narration. We wanted identical narration so that there would only be the difference in whether the memory model was shown and no inadvertent differences in narration. Of course for a specific learning purpose an instructor could provide specific detailed instruction on the memory model when it is shown. The learning materials were focused on preparing the participant for the tracing questions and not the relational response questions. There was no discussion of summarizing or relationally reasoning about the code.

To design the instruction we applied the cognitive load reducing methods of segmenting and weeding to focus on teaching the concepts without all the other material that would be covered early in an introductory programming course. As we will see, we were reasonably successful considering the videos are 5 minutes in length. However, further application of the cognitive load reducing methods for Learning Survey 2

---

[12] Survey 1: instruction without memory shown: https://www.youtube.com/watch?v=FIhThMCrdpI

resulted in a substantial increase in the number of participants successfully answering the tracing questions.

The Qualtrics Survey software has the capability to randomly show one of the two videos. We used this capability to create our two conditions. When a participant had answered the demographic and prior knowledge questions, he/she was randomly shown one of the instruction videos. Which video was shown and how long the video was watched was recorded for each participant.

Survey Questions

As described above, we derived our tracing and relational reasoning questions based on Corney et al (Corney et al., 2011; Teague et al., 2012). In the first survey, in Question 2 the second to last line, "p = q;" was inadvertently removed but was restored in the second and third surveys to be consistent with Corney et al.

Questions 4 and 5 ask about code that swaps the values in two variables utilizing a third variable. Both questions are identical except Question 4 provides an explanation and an example relational response while Question 5 does not. To test how much the participants rely on the explanation and example provided in question 4, we reversed the order that questions 4 and 5 were asked. Question 5 was presented on an online page of its own and then Question 4 was presented on the next page. There was not a

means to return to a previous question after completing it. The complete survey is shown in Appendix C.

Advertisement

We advertised our task on Amazon Mechanical Turk (AMT) as "Learning Survey: Watch a short video and answer some questions". We paid $0.25 per participant and received 499 responses that finished and gave consent. Use of AMT with a similar pay rate has been found quite effective for similar research (Buhrmester, Kwang, & Gosling, 2011; Horton & Chilton, 2010; Lee & Ko, 2011; Paolacci et al., 2010).

Learning Survey 2

In this survey we have the three conditions, instruction with computer memory model, instruction without computer memory model and no instruction.

Learning Materials

For the instruction conditions, we intended to improve on the instructional videos from Learning Survey 1. Since variable-to-variable assignment seemed to be a stumbling point by many in the first learning survey, we created an example that included two lines of code demonstrating variable-to-variable assignment. In the ReadJava simulator we simplified the highlighting of the code (improved signaling) and increased the font size. Also, we were concerned that even at 5 minutes we had covered "too much, too

quickly" so we shortened to 3 minutes. We applied the cognitive load reducing

methods, weeding and segmenting, to shorten the time by abbreviating our discussion

of declaration and assignment of initial values and the discussion related to literals. At

the beginning of the video we added a statement describing the purpose of the example

(pretraining). At the end we mentioned the purpose again. To create the instructional

videos we performed the same procedure as we did for the first learning survey. We

recorded a screencast of the author using the ReadJava simulator with the computer

memory model showing to create the instruction with computer memory model

video[13]. Then we copied the video and visually removed the computer memory model

to create the instruction without computer memory model video[14]. The resulting

Learning Survey 2 videos were 3 minutes long.

To assess how successful participants are without any instruction, we created a

condition with a brief video without instruction[15]. In the 39 second video, we mentioned

we were assessing the prior knowledge that participants bring to programming

instruction and ask that they answer the questions the best they can. We expected a

---

[13] Survey 2: instruction with memory model: https://www.youtube.com/watch?v=P4ZEcolWgk4

[14] Survey 2: instruction without memory model: https://www.youtube.com/watch?v=76RfvVVixsg

[15] Survey 2: instruction video: https://www.youtube.com/watch?v=ufAVwIsX4DE

reasonable effort, similar to the effort for the other condition, since they have chosen to participate and this was the assigned task.

Survey Questions

We asked the same five tracing and relational response questions as in Learning Survey 1 with the following changes. We made question 2 consistent with Corney et al but with our modifications for Java. Questions 1 and 3 are identical to Learning Survey 1. Also, we asked question 4 before question 5 as Corney et al had done, in order to confirm the consistency results in our environment that they had reported. The questions are shown in Appendix C.

Advertisement

The advertisement on Amazon Mechanical Turk changed in four ways from the first learning survey. First, we increased the Reward (pay) to $0.50 to reduce the time to collect the data. Burhmester et al (2011) reported that increased pay essentially results in quicker data collection as low pay did not appear to affect data quality. Secondly, we added that the location must be in the United States to the "Qualifications Required". About 89% of the participants in our first learning survey came from Asia and the United States. We noticed about two-thirds of the Asian participants already had programming knowledge that we were paying for, but our focus was on participants without

programming knowledge. About two-thirds of the participants from the United States, however, did not have programming knowledge. The third difference was that we reduced the estimated duration to 15 minutes rather than 30 minutes. In our first experiment, the average time was less than 8 minutes and we had reduced the length of the video in this second experiment. The task preview changed slightly with the length of the video being described as three minutes rather than five, which is the fourth difference.

Learning Survey 3

In this survey we directly compared the two instructional videos without computer memory[16] from Learning Survey 1 and Learning Survey 2. The survey questions are the exact same questions from Learning Survey 2. The advertisement was identical to Learning Survey 2, with reward (pay) of $0.50, advertised in the United States and estimated duration of 15 minutes. The task preview changed slightly from Learning Survey 2 due to the varied length of the videos. We changed "watching a 3 minute

---

[16] The first learning survey video is accessible via the link:

https://www.youtube.com/watch?v=FIhThMCrdpI

The second learning survey video is available: https://www.youtube.com/watch?v=76RfvVVixsg

video" to "watching a short video" and changed the estimated time to complete from "about 8 minutes" to "about 10 minutes".

Results

Data Processing

After the participants completed each survey, the data was downloaded as a spreadsheet from the Qualtrics Software survey website. The spreadsheet columns were for each question response and additional information such as whether the participant completed the survey, which video the participant watched and the time the participant actually spent watching the video. Each row was a record of one participant's answers to the questions.

We wrote a program in Java to filter and process the data. Within the program, the Apache Commons Math library[17] was used for the statistical calculations. Excel was used to create the charts.

Adding Relational Response Categories

The first step in analyzing the data is to add two columns to the spreadsheet to categorize the answers to the two Relational Response questions. Corney, Lister and

---

[17] http://commons.apache.org/proper/commons-math/userguide/stat.html

Teague defined a relational response as "the [participant] provides a correct summary of the overall computation performed by the entire piece of code." (Corney et al., 2011). Since we were specifically looking for relational responses we defined four categories: 1) correct relational response, 2) partially correct relational response, 3) other, and 4) blank. The 'other' category was the "catch all" for incorrect relational responses, correct and incorrect multi-structural responses, and any other comments or responses that the participant provided. Our Correct Relational Response category seems to be very similar to a correct response as described in the replication paper of the original study (Teague et al., 2012).

- Correct Relational Responses were relational, correct and would receive full credit on a test. Examples include, "it swaps the values in i and k", "This code is use to interchange values of variable i and k" and "interchanging values of i and k with the help of j".

- Partially Correct Relational Responses were relational and on the right track but we thought were too general or had a minor error. As participants may have never been exposed to code that swaps values, they may not know terminology (e.g., the term "swap") to describe it. Examples include "interchange of values", "this is for interchanging the value j to k", "swapping the values among the variables." and "to circularly shift the values in the variables".

- <u>Other</u> responses were relational and incorrect, or not relational, (e.g., multi-structural), or were more comment oriented. Multi-structural responses are those that describe what the separate lines of code are doing, but do not summarize the purpose of the code. Examples include: "The three values are equal....", "To over write their initialized values with other values .", "To compare any variables.", "I have no idea" and "This would replace the value of J with that of I, I with that of K, and K with the NEW value of J."

- The <u>blank</u> category is for responses that are left blank.

Results of Learning Survey 1

Data Gathering

We gathered 765 records during two weeks in December 2013. Our first step was to categorize the relational response questions as described above without looking at any of the other fields in each record.

Data Filtering

Three participants did not give consent, 263 did not finish and 31 participants did not answer "no" to the catch trial leaving 468 records. Since our objective was to assess the effectiveness of the video, we eliminated 208 participants that did not watch the full 5 minutes and 13 seconds. The large majority of those had programming experience and

watched the video for less than 1 minute. Of the 260 remaining records, 137 answered

"yes" to the question "Have you ever had training on how to write a computer program

either in a classroom or online?" while 123 answered "no". Of the 123 participants that

reported not having programming training, 61 were randomly chosen to view the

instruction without memory model video and the remaining 62 were chosen to watch

the instruction with memory model video. Our analysis here is based on these 123

participants.

Demographics

The ages ranged from 18 to 69 with a mean age of 37.8 and a median age of 35.

Approximately 65% held at least a bachelor's degree with only one not being at least a

high school graduate. Approximately 54% were female with 46% male with only one not

choosing to respond to the question. Approximately 45% of the participants came from

the USA and another 44% from Asia.

Analysis of Tracing Questions

Originally we were testing whether there was a significant difference in the number of

correctly answered tracing questions between the two conditions, instruction without

computer memory showing and instruction with computer memory showing. A t-test,

shown in Table 2, indicated there was not a significant difference.

Number of Tracing Questions Correct	Without memory	With memory
n	61	62
mean	1.148	1.355
SD	1.223	1.189
p (significant at 0.05)	0.34 (false)	
Cohen's d	0.17 (small effect)	

Table 2: Learning Survey 1 t-test on Number of Tracing Questions Correct

Next, we looked at the results of each tracing question to see if there were significant

differences.
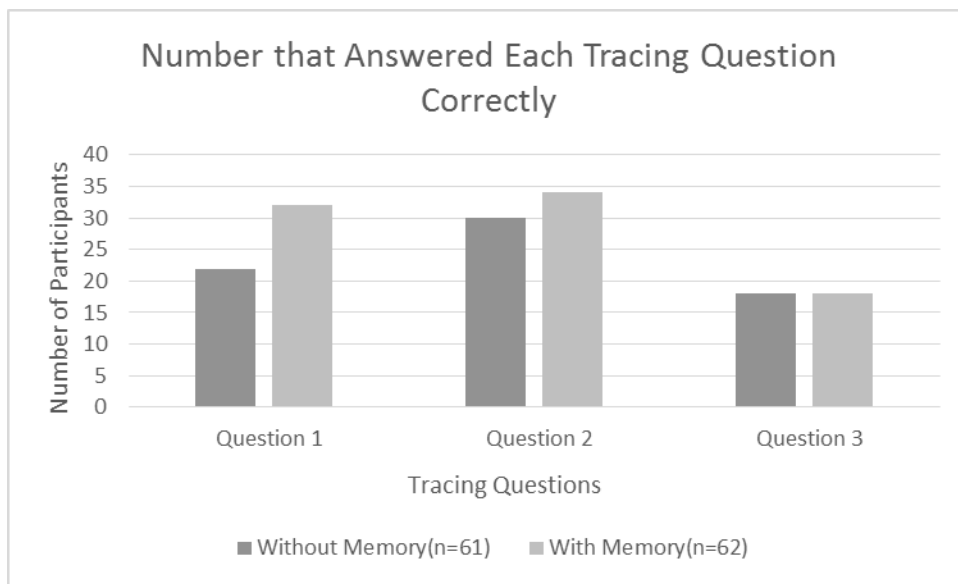
Figure 17: Learning Survey 1 Number that answered each Tracing Question Correctly

Figure 17 shows the number of participants without programming that answered each
of the tracing questions correctly. The chart suggests that the computer memory model
may be useful for some novices, particularly on Question 1. However a t-test results in
p=0.08 which is not statistically significant according to the traditional p=0.05 criterion
(Table 3). The 0.05 criterion means there is a 1 in 20 chance our data has occurred by
random. A p=0.08 means that our data has a 1 in 12 chance of occurring by random.

t-test for each question	Question 1		Question 2		Question 3	
	Without Memory	With Memory	Without Memory	With Memory	Without Memory	With Memory
n	61	62	61	62	61	62
mean	0.36	0.52	0.49	0.55	0.30	0.29
SD	0.48	0.50	0.50	0.50	0.46	0.46
p	0.08 (false)		0.53 (false)		0.95 (false)	
Cohen's d	.33 (small effect size)					

Table 3: Learning Survey 1 t-test for each Tracing Question for non-programmers

Tracing Questions Correctly Answered (Without Memory)	Participants from USA[18] (n=32)	Participants Not from USA (n=29)	Participants Combined (n=61)
Question 1	10 (31%)	12 (41%)	22 (36%)
Question 2[19]	16 (50%)	14 (48%)	30 (49%)
Question 3	6 (19%)	12 (41%)	18 (30%)
All 3 Questions	5 (16%)	9 (31%)	14 (23%)

Table 4: Learning Survey 1 Number of Correct Answers on Tracing Questions for the Without Memory condition

Table 4 shows the number of correct answers on the three tracing questions as well the number of participants that answered all three tracing questions correctly for the instruction without computer memory condition. In Table 4 we separate out the participants from the USA as a comparison to Learning Survey 2 where we only advertise for participants in the USA.

For combined participants, we note that most were not successful answering any of the questions with success rates ranging from 30% to 49%. Only 14 out of 61 (23%) for the combined group answered all three questions correctly. Since the video covered an

[18] We report the data as from USA and Not from USA since in Learning Survey 2 we advertise for participants only from the USA since most participants from the USA do not have programming training which we desire.

[19] In Learning Survey 1, question 2 differed from the original. In Learning Survey 2, question 2 is consistent with the original.

example very similar to the problems, these results seem to confirm that the concepts

of assignment and sequence are challenging for many novices.

To gain insight on potential difficulties, we analyzed the wrong answers provided for the

three questions. The wrong answers for each question that occurred five or more times

are shown in Table 5.

Common Wrong Answers (Without Memory)	Question 1	Question 2	Question 3
Wrong Answer (occurrences)	r=2, s=4 (26) r=s, s=4 (5)	p=1, q=8 (15) p=1, q=p (6)	x=5, y=3, z=7 (12) x=7, y=5, z=3 (11)

Table 5: Learning Survey 1 Common Wrong Answers for Tracing Questions

Our analysis suggests that these wrong answers may potentially be arrived at by simply

guessing or ignoring a line or two of code that one does not understand, such as

variable-to-variable assignment. For example, it does not seem surprising to us that

when a novice is shown the following code fragment from Question 1:

```
r = 2;
s = 4;
r = s;
```

The novice may provide the wrong answers given, when asked for the values of r and s.

The most frequent wrong answer "r=2, s=4" seems to be simply ignoring the "r = s;" line

of code. The second most frequent wrong answer "r=s, s=4" seems to indicate that the

novice does not realize that "s" is a variable that contains a value. Since so many

participants were not successful giving the answer after having presented these

concepts in a very similar example, perhaps we still overloaded participants' cognitive

systems with too much material too quickly. Therefore, questions we have are:

1) Will focusing more explanation on the seemingly difficult concept of variable-to-
   variable assignment be helpful?

2) Can we weed out additional extraneous material to help the participant focus on
   the most difficult material?

Since we did not have a control group that had participants answer the questions

without having instruction, we do not know how effective our question is that asked

about previous programming training or whether the answers could be easily guessed

by someone without instruction. We address this with a no instruction group in

Learning Survey 2.

Analysis of Relational Response Questions

Our instruction was intended to help with the tracing questions but was not intended to

help with the ability to summarize the meaning of a section of code. Interestingly, the

computer memory model had a statistically negative effect (p=0.01, d=0.45) on the

ability of participants to answer question 4 correctly. Perhaps the computer memory

model was powerful enough that participants recalled it and assumed it would be
helpful even though the information within it would not be helpful for summarizing the
purpose of a section of code.
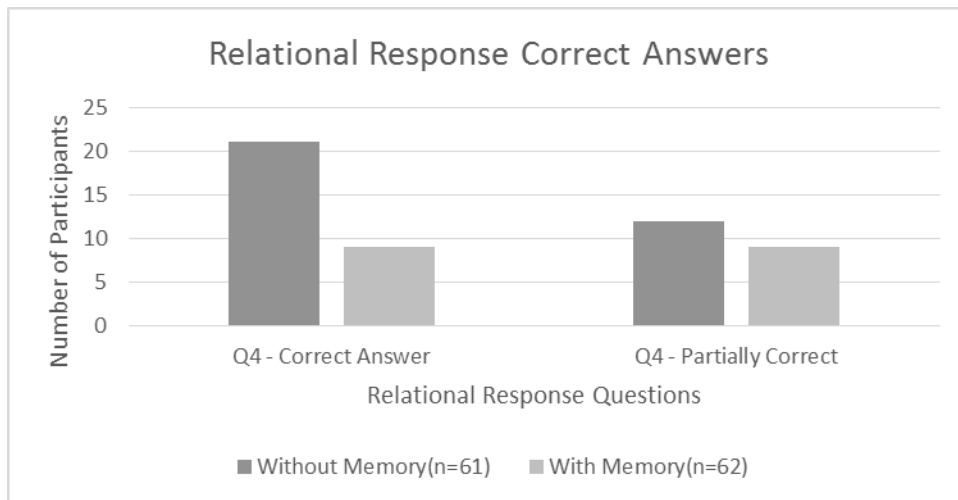
Figure 18: Learning Survey 1 Relational Response Correct Answers

As Corney et al (2011) had done, we reviewed the responses to questions 4 and 5 for
the 14 participants that answered all three tracing questions correctly. We note that 9
of the 14 (approximately half) participants that answered all 3 tracing questions
correctly also correctly answered question 4.

Questions 4 and 5 are the relational response questions. As noted, we asked question 5
and then question 4. Recall that the problems in the questions are identical, however,
question 4 has an explanation and an example that, if the participant recognizes it, is a

solution to the problem. Therefore, keep in mind that in Table 6, question 5 was asked before question 4.

Relational Response when all 3 tracing questions answered correctly (Without Memory) (n=14)	Correct Answer (from USA)	Correct Answer (not from USA)	Partially Correct (from USA)	Partially Correct (not from USA)
Question 4	4	4	0	4
Question 5	0	1	0	6

Table 6: Learning Survey 1 Number of Correct Answers for Relational Response Questions

We note that only 1 out of 14 participants were able to answer the question 5 correctly without having the additional explanation and example solution provided in question 4. On the other hand, 8 out of 14 were able to answer question 4 correctly after seeing the explanation and example solution. If we include the partially correct answers, then only 7 out of 14 were able to answer question 5 correctly, while 12 out of 14 were able to after seeing the explanation and example solution.

Since the problems in the two questions are the same, we conclude that the explanation and example solution at the beginning of question 4 are very important for assisting participants in understanding the solution we are looking for as well as providing an explanation of the solution. As noted previously, the example solution is the solution to the problem, if the participant recognizes it.

Results of Learning Survey 2

Data Gathering

We offered the second learning survey on Amazon Mechanical Turk in March 2014. It took 4 days for 413 participants to start the survey. The average time to watch a video and complete the survey was 6 minutes and 55 seconds.

Workers on AMT are identified by a unique id and so we can tell if they have worked for us previously. Out of the 338 participants that completed the task on AMT, 6 had also taken Learning Survey 1 offered over three months earlier. However, for anonymity purposes we did not track which participant submitted each survey and so we do not know the group from either experiment each participant was in or whether a participant had indicated they had programming training in either experiment. However, given the small number and the random assignment to groups, we do not believe these 6 participants would have a significant effect on the results.

We categorized the responses to the relational response questions the same way as in Learning Survey 1. The four categories are Correct Relational Response, Partially Correct Relational Response, Other and Blank. We adapted our Java program to process the results for this new set of data.

Data Filtering

One participant did not give consent, 68 did not finish, and one failed the catch trial question leaving 343 records. Only 25 of the 343 did not watch the full length of the video, leaving 318 records. We separated out the 98 that reported having programming training leaving 220 records. Of those 220 without programming training, 72 were in the instruction without computer memory condition, 72 were in the instruction with computer memory condition and 76 were in the no instruction condition.

Demographics

The ages ranged from 18 to 71 with a mean age of 35.7 and a median age of 32. Approximately 45% held at least a Bachelor's degree with all but two participants at least a high school graduate. About 54% of the participants were female and 46% male. All but two participants[20] were from the United States.

---

[20] While we asked Amazon Mechanical Turk to advertise only to participants in the United States, one participant responded to the "Where are you located?" question on the survey with "Africa" and the other "South America". Perhaps they were simply overseas at the time of participating. In any case, our focus was on finding participants that did not report a programming background.

Analysis of Tracing Questions

When comparing the number of correctly answered tracing questions between the two instruction conditions there was not a statistically significant difference (p=0.27). Interestingly, the number of participants who answered Question 1 correctly was significantly different between the two instruction conditions (p=0.03,d=0.36). There was a small effect size, but in contrast to Learning Survey 1, the instruction without the computer memory model was more helpful to the participants. The number of correct answers in Questions 2 and 3 were not significantly different. Figure 19 shows a chart comparing the differences.
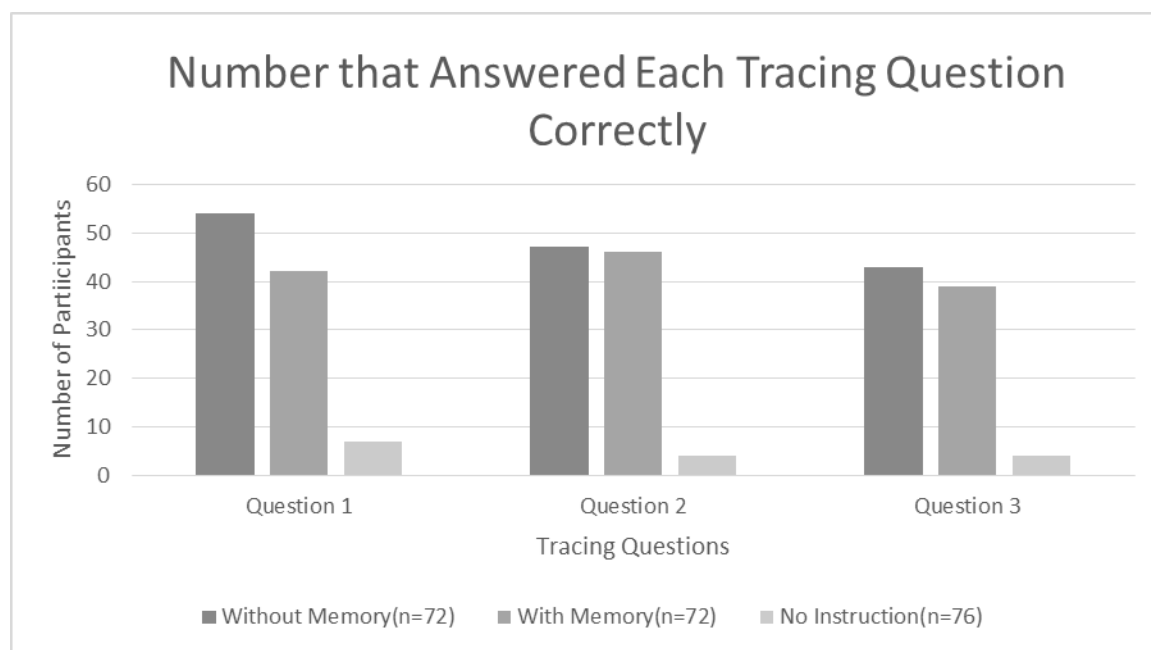
Figure 19: Learning Survey 2 Number that answered each Tracing Question Correctly

Table 7 shows the number of participants that answered each tracing question correctly as well as the number that answered all three questions correctly. In the no instruction condition only 3 out of 76 (4%) answered all three tracing questions correctly. That so few participants were successful without instruction seems to indicate that our question asking about prior programming training is successful in separating out the participants without programming training. This result also seems to indicate that only a small percentage of participants have some ability or prior knowledge that enables them to be successful without instruction.

In comparison, 41 out of 72 (57%) in the instruction without computer memory condition and 32 out of 72 (44%) in the instruction with computer memory condition answered all three tracing questions correctly. Therefore, we are confident the video instruction we are providing has a strong effect with the participants in our study.

Tracing Questions Correctly Answered	Instruction Without Memory Condition All from USA but 1 (n=72)	Instruction With Memory Condition All from USA (n=72)	No Instruction Condition All from USA but 1 (n=76)
Question 1	54	42	7
Question 2	47 (65%)	46	4
Question 3	43	39	4
All 3 Questions	41 (57%)	32 (44%)	3 (4%)

Table 7: Learning Survey 2 Number of Correct Answers on Tracing Questions

Analysis of Relational Response Questions
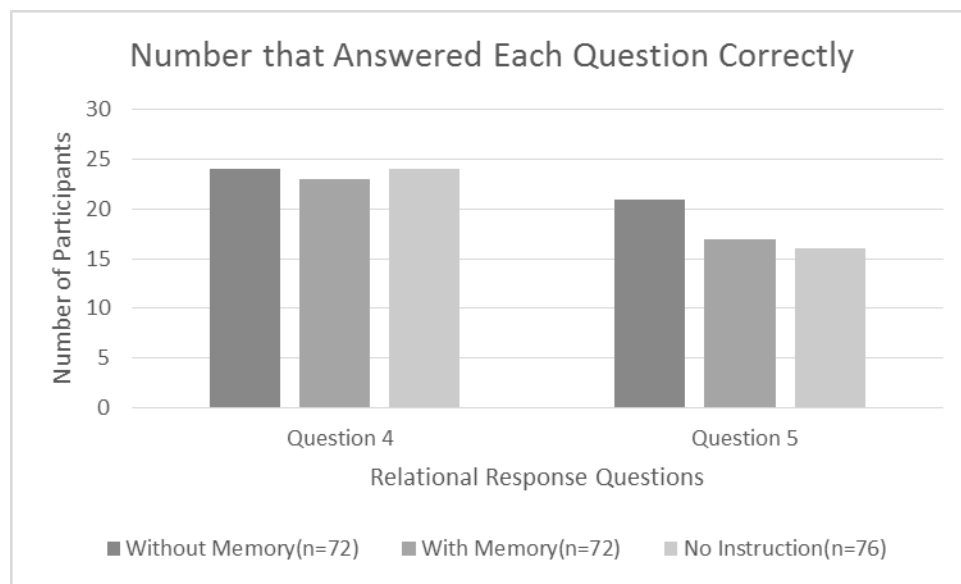
Figure 20: Learning Survey 2 Relational Response Correct Answers

As shown in Figure 20, for all the participants in the instructional conditions there was

not a significant difference in the number of correct answers on the relational response

question 4. Therefore the computer memory model seems to not be helpful or hurtful

for these particular questions. More interesting though is the No Instruction group performed as well as the instructional groups seemingly indicating that the instruction was not helpful for these questions and similarly the questions are not measuring the effectiveness of the instruction.

In Table 8 the results are for those participants on questions 4 and 5 that had answered all three tracing questions correctly. There were 41 in the instruction condition, but only three in the no instruction condition that answered all the tracing questions correctly. Recall that in this Learning Survey 2, question 4 was asked before question 5, therefore the explanation of a correct response and the example answer were seen by participants before answering either question.

Relational Response Question Correctly Answered for Participants with all 3 Tracing Questions Correct	Instruction Without Memory Condition (n=41)	Instruction With Memory Condition (n=32)	No Instruction Condition (n=3)
Question 4 Correct	20 (49%)	13 (41%)	2 (67%)
Question 5 Correct	19 (46%)	11 (34%)	2 (67%)
Question 4 Partially Correct	6 (15%)	4 (13%)	0 (0%)
Question 5 Partially Correct	3 (7%)	5 (16%)	0 (0%)

Table 8: Learning Survey 2 Number of Correct Answers for Relational Response Questions

In the no instruction condition, we notice that 2 out of 3 participants answered both questions correctly. This seems to indicate prior knowledge or ability for these participants. In the instruction condition, 20 out of 41 (49%) answered question 4 correctly and 19 out of 41 (46%) answered question 5 correctly. There were 17 of the 20 (85%) who answered question 4 correctly and also answered question 5 correctly.

Results of Learning Survey 3

Data Gathering

We offered the third learning survey on Amazon Mechanical Turk in April and May 2014. It took 7 days for 654 participants to start the survey. The average time to watch a video and complete the survey was 8 minutes and 25 seconds.

As described above, a unique id identifies workers on AMT so we can tell if they have previously worked for us. Out of the 500 participants that completed the task on AMT, four had previously taken Learning Survey 1, 49 had previously taken Learning Survey 2 and two additional had taken both Learning Survey 1 and 2 for a total of 55 out of 500 (11%) that were repeat workers. However, for anonymity purposes we did not track which participant submitted each survey and so we do not know the group from any experiment each participant was in or whether a participant had indicated they had programming training in either experiment. However, given the random assignment to conditions, we do not believe the 11% of participants would have a substantial effect on the results.

We categorized the responses to the relational response questions the same way as in Learning Survey 1 and 2.

Data Filtering

Four participants did not give consent, 129 did not finish, one failed the catch trial question and 68 did not watch the full length of the video, leaving 452 records. We separated out the 130 that reported having programming training leaving 322 records. Of those 322 without programming training, 167 watched the instructional video from

Learning Survey 1 and 155 watched the instructional video from Learning Survey 2. We

report on the 322 without programming training below.

Demographics

In the LS1 instructional video condition, ages ranged from 18 to 69 with a mean age of

35.9 and a median of 33. In the LS2 instructional video condition, ages ranged from 18

to 73 with a mean of 36.8 and a median of 33. All but two participants were high school

graduates. 49.7% of the LS1 instructional condition held at least a bachelor's degree,

while 51% of the LS2 instructional condition held at least a bachelor's degree. In the LS1

instructional condition, 60.5% of participants were female while 68.4% in the LS2

instructional condition were female. Almost all, 318 out of 322, of the participants were

from the United States.

Analysis of Tracing Questions

This survey, Learning Survey 3, was to confirm the differences between the two videos

with a direct comparison. As shown in Table 9, there was a significant difference with

medium effect size (Cohen's d between 0.40 to 0.47) between the two conditions for

each of the questions individually as well as the sum of correct answers for all three

questions ($p < 0.01$).

Tracing Questions Correctly Answered	LS1 instructional condition (5 min 13 second video) (n=167)	LS2 instructional condition (3 minute video) (n=155)	t-test
Question 1	68	96	p<0.01, d=0.43
Question 2	59 (35%)	90 (58%)	p<0.01, d=0.47
Question 3	55	81	p<0.01, d=0.40
All 3 Questions	48 (29%)	69 (45%)	

Table 9: Learning Survey 3 Number of Correct Answers on Tracing Questions

Analysis of Relational Response Questions

Table 10 shows the number of correct answers on questions 4 and 5 for those

participants that answered all three tracing questions correctly. The results in this

survey are consistent with our previous surveys with about half of the participants

answering questions 4 and 5 correctly.

Relational Response Question Correctly Answered for Participants with all 3 Tracing Questions Correct	LS1 instructional condition (5 min 13 second video) (n=48)	LS2 instructional condition (3 minute video) (n=69)
Question 4 Correct	29 (60%)	36 (52%)
Question 5 Correct	24 (50%)	33 (48%)
Question 4 Partially Correct	4	8
Question 5 Partially Correct	9	4

Table 10: Learning Survey 3 Number of Correct Answers for Relational Response Questions

Discussion

Threats to Validity

Our study has a number of threats to its validity and generalizability. Amazon

Mechanical Turk (AMT) allows participants to self-select into tasks if they meet

qualifications, in our case a 95% HIT approval rate from previous requestors. We tried

to account for factors that would affect the task listing such as a title and description

that provide an idea of the task (e.g., Learning Survey) but did not mention the content

was programming. We did not want potential participants to self-select based on the

content. However, as workers on AMT can preview tasks, and in fact many did start the

survey but did not finish, some self-selection based on content seemed to occur.

Significant computer use knowledge is necessary as this assessment and video are

shared via an internet connected computer that require the user to have an account

and login. There is an economic incentive for participants to participate in our study.

Although small, this incentive does not exist in a classroom and so may have an effect.

There was not a specific incentive provided for correct answers while in a classroom

situation, a better grade is an incentive. Also, users of AMT have an option to

participate whereas students in a classroom frequently may not.

The author created both scripts and videos and this research was conducted as the

author believed that these cognitive load reducing methods would likely be helpful for

introducing introductory programming concepts. We made a number of changes,

consistent with reducing cognitive load, from the first to the second video, some of

which may be more influential than others in the results.

Teaching to the Test

One critique is that we are teaching to the test. We are providing instruction for very

similar problems that we ask the participants to solve. To us, this is not simple recall of

presented material, but is a transfer test, that requires the material to be learned and

applied to new problems. We note that in Learning Survey 1 only 23% were able to

successfully apply the presented material to solve all three problems even though the

teaching example was very similar to the test. In Learning Survey 2, only 57% were able

to solve all three problems. Therefore this suggests that the three tracing questions constitute a transfer test that reflects meaningful learning.

In our mathematics education, we received instruction on how to solve a problem and then had to solve a set of similar problems at the end of each section. In each section, we built skill with lots of practice before going on to more challenging problems that relied upon previous skills.

Pedagogical Implications

The first learning survey video was, what we thought when creating it, a reasonable presentation of the declaration, assignment and sequence concepts applying cognitive load reducing methods. With the 5 minute and 13 second video, 23% of the participants were able to successfully answer all three questions. However, after analyzing the actual results and weeding, segmenting, improved signaling and presenting in a 3 minute video, 57% of the participants were able to successfully answer all three questions. That cognitive load reducing methods are effective is well documented (Mayer & Moreno, 2003). That these methods are very effective for teaching introductory programming concepts that are perceived as difficult for many novices is interesting. How we instruct novices in introductory programming concepts matters a great deal and is anticipated by the Learning Edge Momentum hypothesis.

Porter and Zingaro (2014) study provided evidence that the reason that success with

fundamental concepts early in a course is related to success in the course is due to

those fundamental concepts being a part of most questions on a final exam. They

"…recommend that instructors pay extra attention to those [fundamental course]

concepts, perhaps performing early interventions with students who demonstrate

relevant misconceptions." In Learning Survey 2 we note that 17 out of 72 (24%) were

still not able to solve any of the tracing problems. Since our participants only spent 3

minutes on the content, we are optimistic that there is more room for improvement of

our instruction. Perhaps having supporting materials such as simulations and videos

applying the cognitive load reducing methods along with multiple problems that give

feedback may help many students. Intelligent tutoring systems may also be helpful as

they have been shown to be "nearly as effective as human tutoring" (VanLehn, 2011).

Theories on the Difficulty with Learning Programming

Ahadi & Lister (2013) discuss stumbling points, prior knowledge, and "geek genes" along

with Learning Edge Momentum theory (Robins, 2010) about why some students thrive

and others struggle to learn to program. The 4% of participants in the no instruction

condition of our Learning Survey 2, that reported they had not had programming

training and still managed to answer all three tracing questions correctly, seem to

support the theories that there is prior knowledge or ability for a small percentage of participants.

The 57% of participants that were able to successfully answer the three tracing questions in the instruction without computer memory condition in the second Learning Survey and the 24% that were not able to answer any of the tracing questions correctly seems to support the stumbling point hypothesis.

However, the significant gain from 23% from the instruction without computer memory condition in the first learning survey that we thought was a reasonable presentation to 57% in the same condition in the second learning survey seems to lend strong support to the hypothesis that our typical methods of instruction simply overwhelm the cognitive capabilities of the students. As Robins (2010) notes, the mastery model of learning that other fields have learned long ago may be useful for us to apply in teaching introductory programming.

Computer Memory Model

That the number of correctly answered tracing questions between the instruction conditions was not significantly different for either the first or second learning survey seem to indicate that the computer memory model does not make a difference overall for the specific concepts taught here. However, when we looked at individual questions

we may be seeing some effect that is worthwhile to investigate further. In the first learning survey the difference in the instruction conditions for question 1 was almost significant (p=0.08) with the computer memory model seeming to be potentially helpful for some participants. However, in the second learning survey the difference in the instruction conditions for question 1 was significant (p=0.03) but, in contrast to the first survey, the computer memory model was not helpful. In both the first and second learning surveys the difference in number of correct answers between the instruction conditions for questions 2 and 3 were not significant. Since the effect of the computer memory model is less for questions 2 and 3, perhaps this indicates the effect is very short-lived. Or perhaps there is something different about the first question than the others.

For the relational response question 4, in Learning Survey 1 there was a significant negative effect for those with the computer memory model. However, in Learning Survey 2 there was virtually no difference at all. Perhaps when the instruction was more confusing than the computer memory model was helpful for some participants to utilize to help themselves learn to trace. But since the computer memory model emphasizes details and not summary of purpose it was not helpful, and perhaps misleading, for some participants when asked to summarize the purpose of code.

Ben-Ari asserts "students do not have an effective model of a computer" and "models must be explicitly taught"(Ben-Ari, 1998). Mayer, in his study of how novices learn computer programming concurs that "a concrete model can have a strong effect on the encoding and use of new technical information by novices" (Mayer, 1981). Contrastingly, Petre notes that while visualizations are appealing, they are not necessarily helpful, usually slower to acquire the same information and frequently require learning a secondary notation regarding layout, typographic cues and graphics (Petre, 1995). For Computer Science Education specifically, Holzinger et al review factors that contribute to the success of static or dynamic media. They found:

> "Dynamic media is only successful in facilitating learning in comparison to traditional static media such as texts or images, when they are able to (1) reduce the cognitive load, which is necessary to comprehend them, (2) serve to generate mental models of a concept and, consequently (3), offer visualizations that *correspond to a meaningful mental model*. [original emphasis]

> "Moreover, dynamic media must be attuned to learners' experience, expertise, and most of all previous knowledge. Therefore, material containing dynamic media must avoid information, animations, and elements, which are not necessary to comprehend a concept." (Holzinger, Kickmeier-rust, & Albert, 2008)

We had thought we were taking this advice into account when designing the computer memory model. More investigation in the appropriate use of visualizations and models seems warranted.

Reading Comprehension

There are many differences in our learning experiments and those reported by Corney et al (2011) and Teague et al (2012) such as:

- Ours is a very short study with many demographically diverse participants that already have at least a bachelor's degree versus traditional students that are earning a college degree.

- Ours is short term and presents very little material and then tests participants immediately versus a traditional classroom course that covers a lot of material over a long period of time and tests participants over time.

However, these differences make similarities in results potentially interesting. One similarity is that approximately half of the participants in each of our learning surveys that watched the instructional videos without the computer memory and answered all three tracing questions correctly also answered question 4 correctly. Corney et al noted a similar relationship (39 out of 83) and reported that those that were successful in week 3 on question 4 also had a very high rate of success in the course.

The consistency in results (around half successful on question 4) seem to indicate that the different instructional methods used by Corney et al and by us are not a significant factor for the success of students when answering question 4. Learning Survey 1 seemed to show that the explanation and the example solution to the problem provided in question 4 seem to be very important for participants to answer the question correctly. Learning Survey 2, with the no instruction condition performing as well as the instruction conditions on question 4 seems to indicate that the content instruction makes no difference for success on the question. Perhaps participants that are not successful with answering question 4 are not successful simply because they do not comprehend that the solution is presented to them. Not comprehending what is read is a side effect of not reading proficiently.

> "Proficient reading depends on the ability to recognize words quickly and effortlessly (Adams, 1994). If word recognition is difficult, students use too much of their processing capacity to read individual words, which interferes with their ability to comprehend what is read." [21]

A potential implication is that many of those students that are not successful on question 4 and that do not learn in a traditional lecture are not successful on the final

---

[21] http://en.wikipedia.org/wiki/Reading_comprehension retrieved June 4, 2013.

exam because of difficulty with comprehending written explanations such as those in a

textbook. Use of the cognitive load reducing methods, specifically multimedia, may be

very helpful for these students. We can imagine utilizing cognitive load reducing

methods to supplement existing materials or to create a new kind of textbook.

5. Conclusion

We have utilized non-traditional methods to gather new data on the teaching and learning of introductory programming. For the design of instruction of fundamental programming concepts, we have utilized cognitive load reducing methods with strong data support (Mayer & Moreno, 2003). Our data seems to suggest how we teach these fundamental programming concepts matters a great deal which is anticipated by the Learning Edge Momentum theory (Robins, 2010). In summary, utilizing cognitive load reducing methods for instruction seems to have significant potential to assist many people with learning fundamental computer programming.

On the other hand we had anticipated that our simple computer memory model would have a consistently positive effect which turned out not to be the case. We need more investigation on how to effectively design and when to effectively utilize a memory model.

The author and narrator of the videos, while having had the opportunity to lecture for many hundreds of hours, never before had the opportunity to prepare 5 minutes of instruction with a particular performance learning goal, measure the results of the 5 minutes in terms of actual performance of a significant number of participants and then design additional instruction to improve and assess the differences. This has been eye

opening in terms of the potential impact of being very aware of content specific learning

challenges and carefully designing instruction following cognitive load reducing methods

to assist students with their learning.

As we have utilized Corney et al's (2011) materials, at their urging in their paper, to

study our own instruction we too encourage the reader to consider utilizing our

methods or videos to help assess whether our results are unusual or more widespread.

What Was Done

We developed hypotheses regarding whether cognitive load reducing methods and a

simple computer memory model would be helpful for novices to learn the meaning of

some introductory programming concepts. We built a Java program simulator with the

computer memory model and developed examples illustrating declaration, assignment,

sequence, conditional, looping, method calls, recursion, and class instantiation.

We developed learning units by recording screencasts of the author using the ReadJava

simulator. We selected published questions to test our memory model for the

declaration, assignment and sequence concepts and ran experiments comparing a

version of the learning unit with the computer memory model to the learning unit

without the computer memory model. Our Learning Survey 1 results suggested that we

attempted "too much too quickly" in our 5 minute learning unit so revision and reassessing might be worthwhile.

We revised our learning units and reassessed. We discovered that using the cognitive load reducing methods seemed to have a large impact but that the results for the memory model were mixed.

Why It Was Good

To our knowledge, no one has built a complete learning unit that contains both a simplified computer memory model and programming instruction applying cognitive load reducing methods such that there is no learning curve or burden on an instructor or student other than the content itself. We have shown that the simulator is capable of representing in the computer memory model important and difficult introductory programming concepts such as assignment, conditionals, loops, method calls, recursion and class instantiation.

We have gathered empirical evidence that confirms the difficulty of the declaration, assignment and sequence concepts for novices. We also have evidence that shows that many participants are able to learn these concepts with minimal lecture time if the lecture effectively utilizes cognitive load reducing methods.

Future Work

Some specific directions to improve our work include:

1) Study novices using videos of a narrator utilizing our ReadJava simulator on other important and difficult introductory programming concepts.

2) Study novices utilizing the simulator to determine when, where and how to adapt the simulator to make it more effective for novices learning specific concepts.

3) Explore adding interactivity of some kind to the learning units (screen casts of the simulator) in order to provide feedback to participants to help them test their understanding of the concepts.

References

Adams, M. M. (1994). *Beginning to read: thinking and learning about print*. Cambridge, Mass: MIT Press.

Ahadi, A., & Lister, R. (2013). Geek genes, prior knowledge, stumbling points and learning edge momentum: parts of the one elephant? In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (ICER 2013)* (pp. 123–128). ACM.

Anderson, J. R. (1993). *Rules of the Mind* (1st ed.). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Inc.

Ben-Ari, M. (1998). Constructivism in computer science education. In D. J. and J. Impagliazzo (Ed.), *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98)* (pp. 257–261). ACM, New York, NY, USA. doi:10.1145/273133.274308

Bruce-Lockhart, M. P., & Norvell, T. S. (2007). Developing mental models of computer programming interactively via the web. In *Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports, (FIE'07)* (pp. S3H–3–S3H–8). IEEE. doi:10.1109/FIE.2007.4418051

Buhrmester, M., Kwang, T., & Gosling, S. D. (2011). Amazon's Mechanical Turk: A New Source of Inexpensive, Yet High-Quality, Data? *Perspectives on Psychological Science*, *6*(1), 3–5. doi:10.1177/1745691610393980

Clark, R. C., & Mayer, R. E. (2011). *E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning* (3rd ed.). John Wiley & Sons, Inc.

Corney, M., Lister, R., & Teague, D. (2011). Early relational reasoning and the novice programmer: swapping as the hello world of relational reasoning. In *Proceedings of the Thirteenth Australasian Computing Education Conference* (Vol. 114, pp. 95–104). Australian Computer Society, Inc.

Du Boulay, B. (1989). Some Difficulties of Learning to Program. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 283–299). Hillsdale, New Jersey: Lawrence Erlbaum.

Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M. C., & Zilles, C. (2008). Identifying important and difficult concepts in introductory computing courses using a delphi process. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education (SIGCSE '08)* (pp. 256–260). ACM, New York, NY, USA. doi:10.1145/1352135.1352226

Henriksen, P., & Kölling, M. (2004). Greenfoot: Combining object visualisation with interaction. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (OOPSLA '04)* (pp. 73–82). ACM, New York, NY, USA. doi:10.1145/1028664.1028701

Holzinger, A., Kickmeier-rust, M., & Albert, D. (2008). Dynamic Media in Computer Science Education ; Content Complexity and Learning Performance : Is Less More ?, *4522*, 279–290.

Horton, J. J., & Chilton, L. B. (2010). The labor economics of paid crowdsourcing. In *Proceedings of the 11th ACM conference on Electronic commerce (EC '10)* (pp. 209–218). ACM, New York, NY, USA. doi:10.1145/1807342.1807376

Kelleher, C., & Pausch, R. (2005). Lowering the Barriers to Programming : a survey of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, *37*(2), 83–137.

Kelleher, C., & Pausch, R. (2007). Using storytelling to motivate programming. *Communications of the ACM*, *50*(7), 58–64. doi:10.1145/1272516.1272540

Lee, M. J., & Ko, A. J. (2011). Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the seventh international workshop on Computing education research (ICER '11)* (pp. 109–116). ACM, New York, NY, USA. doi:10.1145/2016911.2016934

Levy, R. B. B., & Ben-Ari, M. (2007). We work so hard and they don't use it: acceptance of software tools by teachers. In *Proceedings of the 12th annual SIGCSE conference*

on Innovation and technology in computer science education (ITiCSE '07) (pp. 246–250). ACM, New York, NY, USA. doi:10.1145/1268784.1268856

Lister, R. (2011). Programming, syntax and cognitive load (part 2). *ACM Inroads*, *2*(3), 16–17. doi:10.1145/2003616.2003622

Lister, R., Adams, E., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., … Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. In *Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '04)* (pp. 119–150). doi:10.1145/1044550.1041673

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. In *Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '04)* (Vol. 36, pp. 119–150). ACM, New York, NY, USA. doi:10.1145/1044550.1041673

Mayer, R. E. (1981). The Psychology of How Novices Learn Computer Programming. *ACM Computing Surveys*, *13*(1), 121–141. doi:10.1145/356835.356841

Mayer, R. E. (1997). Multimedia learning: Are we asking the right questions? *Educational Psychologist*, *32*(1), 1–19. doi:10.1207/s15326985ep3201_1

Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning? The case for guided methods of instruction. *The American Psychologist*, *59*(1), 14–9. doi:10.1037/0003-066X.59.1.14

Mayer, R. E., & Moreno, R. (2003). Nine Ways to Reduce Cognitive Load in Multimedia Learning. *Educational Psychologist*, *38*(1), 43–52. doi:10.1207/S15326985EP3801_6

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B. D., … Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, *33*(4), 125–180.

McGettrick, A., Boyle, R., Ibbett, R., LLoyd, J., Lovegrove, G., & Mander, K. (2005). Grand Challenges in Computing: Education--A Summary. *The Computer Journal*, *48*(1), 42–48.

Moreno, A., Myller, N., Sutinen, E., & Ben-Ari, M. (2004). Visualizing programs with Jeliot 3. In *Proceedings of the working conference on Advanced visual interfaces (AVI '04)* (pp. 373–376). ACM, New York, NY, USA. doi:10.1145/989863.989928

Naps, T., Cooper, S., Koldehofe, B., Leska, C., Rößling, G., Dann, W., … McNally, M. (2003). Evaluating the educational impact of visualization. In D. Finkel (Ed.), *Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '03)* (pp. 124–136). ACM, New York, NY, USA. doi:10.1145/960492.960540

Paolacci, G., Chandler, J., & Ipeirotis, P. (2010). Running experiments on amazon mechanical turk. *Judgment and Decision Making*, *5*(5), 411–419.

Parr, T. (2011). *Language Implementation Patterns*. The Pragmatic Bookshelf. Retrieved from http://pragprog.com/book/tpdsl/language-implementation-patterns

Parr, T. (2012). *The Definitive ANTLR 4 Reference*. The Pragmatic Programmers, LLC. Retrieved from http://pragprog.com/book/tpantlr2/the-definitive-antlr-4-reference

Pea, R. D. (1986). Language-Independent Conceptual "Bugs" in Novice Programming. *Journal of Educational Computing Research*, *2*(1), 25–36. doi:10.2190/689T-1R2A-X4W4-29J2

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., … Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin*, *39*(4), 204–223. doi:10.1145/1345375.1345441

Petre, M. (1995). Why looking isn't always seeing: readership skills and graphical programming. *Communications of the ACM*, *38*(6), 33–44. doi:10.1145/203241.203251

Porter, L., & Zingaro, D. (2014). Importance of Early Performance in CS1 : Two Conflicting Assessment Stories. In *Proceedings of the 45th ACM technical symposium on Computer science education (SIGCSE '14)* (pp. 295–300). ACM, New York, NY, USA. doi:10.1145/2538862.2538912

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Yasmin Kafai. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60–67. doi:10.1145/1592761.1592779

Robins, A. (2010). Learning edge momentum: a new account of outcomes in CS1. *Computer Science Education*, *20*(1), 37–71.

Sorva, J. (2012). *Visual Program Simulation in Introductory Programming Education* (No. 0). Aalto University, Espoo, Finland. Retrieved from http://lib.tkk.fi/Diss/2012/isbn9789526046266/isbn9789526046266.pdf

Sorva, J., & Sirkiä, T. (2010). UUhistle – A Software Tool for Visual Program Simulation. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10)* (pp. 49–54). ACM, New York, NY, USA. doi:10.1145/1930464.1930471

Teague, D. M., Corney, M. W., Ahadi, A., & Lister, R. (2012). Swapping as the "Hello World" of relational reasoning: replications, reflections and extensions. In A. de Raadt, Michael & Carbone (Ed.), *Proceedings of the Fourteenth Australasian Computing Education Conference (ACE 2012)* (Vol. 123, pp. 87–94). Australian Computer Society, Inc.

Tew, A. E., & Guzdial, M. (2010). Developing a validated assessment of fundamental CS1 concepts. *SIGCSE*, 97–101. Retrieved from http://dl.acm.org/citation.cfm?id=1734297

VanLehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, *46*(4), 197–221.

Venables, A., Tan, G., & Lister, R. (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. In *Proceedings of the fifth international workshop on Computing education research workshop (ICER '09)* (pp. 117–128). ACM, New York, NY, USA. doi:10.1145/1584322.1584336

Webber, A. B. (1996). The Pascal Trainer. *ACM SIGCSE Bulletin*, *28*(1), 261–265. doi:10.1145/236462.236552

Appendix A: Important and Difficult Programming Topics

ID	Topic	Importance	Difficulty
	Procedural Programming		
PA1	1. Call by Ref. vs Call by Value	7.0 (2.5)	7.4 (1.2)
PA2	2. Formal vs. Actual Parameters	8.6 (1.2)	5.7 (1.7)
PA3	**3. Parameter scope, use in design**	**9.1 (0.9)**	**7.5 (1.0)**
PROC	**4. Procedure design**	**9.8 (0.4)**	**9.1 (0.8)**
CF	5. Tracing Control Flow thru program execution	9.8 (0.4)	7.0 (0.6)
TYP	6. Choosing/Reasoning about data types	8.2 (1.5)	6.6 (0.5)
BOOL	7. Construct/evaluate Boolean expressions	9.5 (0.7)	7.1 (0.8)
COND	8. Writing expressions for conditionals	9.5 (0.5)	6.7 (0.6)
SVS	9. Syntax vs. Semantics	8.6 (0.7)	7.5 (0.5)
OP	10. Operator Precedence	7.1 (1.5)	4.4 (0.5)
AS	11. Assignment Statements	9.5 (1.2)	4.4 (0.5)
SCO	**12. Issues of Scope, local vs. global**	**9.4 (0.7)**	**8.0 (0.0)**
	Object Oriented Programming		
CO	13. Difference between Classes and Objects	10.0 (0.0)	6.9 (1.4)
SCDE	14. Scope design (e.g., public vs private fields)	9.4 (0.7)	6.6 (0.5)
INH	**15. Inheritance**	**7.6 (1.7)**	**9.5 (0.5)**
POLY	16. Polymorphism	7.1 (1.4)	8.9 (0.8)
STAM	17. Static fields and methods	5.7 (1.3)	7.3 (0.6)
PVR	18. Primitive vs Reference variables	8.5 (2.4)	7.0 (0.8)
	Algorithm Design		
APR	**19. Abstraction/Pattern recognition and use**	**8.8 (0.4)**	**9.0 (0.4)**
IT1	20. Tracing nested loop execution correctly	9.5 (0.5)	6.6 (0.7)
IT2	21. Understanding loop variable scope	8.7 (2.0)	4.3 (0.9)
REC	**22. Recursion, tracing and designing**	**7.8 (2.4)**	**9.2 (0.9)**
AR1	23. Identifying off by one index errors	8.9 (0.8)	5.3 (0.5)
AR2	24. Reference to array vs array element	8.4 (1.4)	5.7 (0.7)
AR3	25. Declaring and manipulating arrays	9.0 (1.4)	5.5 (0.5)
MMR	**26. Memory model, references , pointers**	**7.5 (1.7)**	**8.9 (0.7)**
	Program Design		
DPS1	**27. Functional decomposition, modularization**	**9.3 (0.6)**	**7.9 (0.8)**
DPS2	**28. Conceptualize problems, design solutions**	**9.5 (0.5)**	**8.5 (0.5)**
DEH	**29. Debugging, Exception Handling**	**9.0 (0.0)**	**8.6 (0.5)**
IVI	30. Interface vs Implementation	8.1 (0.8)	7.5 (0.5)
IAC	31. Desiging Interfaces, Abstract Classes	5.0 (1.1)	8.6 (0.7)
DT	**32. Designing Tests**	**9.3 (0.8)**	**8.4 (0.8)**

Figure 21: Programming Fundamentals topics rated for importance and difficulty (Goldman et al., 2008).

Figure 21 shows programming topics rated for importance and difficulty. Topics highlighted with **Bold** are the top 11 most important and difficult as rated by Goldman et al. The shaded items are the topics that are directly related to accessing data in computer memory and thus the learner needs to develop an effective model.

1. (PA1) Call by Ref. vs Call by Value
    o   Parameters are variables and this is related to passing a value versus passing a reference that can be used to access values.
2. (PA2) Formal vs. Actual Parameters
    o   Parameters are variables and this is whether we are declaring them as a part of a method definition or passing specific values in a call to a method.
3. **(PA3) Parameter scope, use in design**
    o   Parameters are variables and scope refers to where in the code the variables can be accessed.
4. (CF) Tracing Control Flow thru program execution
    o   Tracing code is dependent upon accessing variable content to make decision about branches to take (conditions), whether to repeat code (loops) or which method call to make (polymorphism).
5. (TYP) Choosing/Reasoning about data types
    o   Variables in Java must be declared prior to use and the declaration specifies a type of data the variable will contain.
6. (AS) Assignment Statements
    o   Save values to variables.
7. **(SCO) Issues of Scope, local vs. global**
    o   Scope is which variables can be accessed from which code.
8. (CO) Difference between Classes and Objects
    o   Class variables have memory allocated when the class is loaded into the JVM while Object variable have memory allocated when the Object (instance) of a class is created. The static keyword is used to differentiate Class variables and methods from Object variables and methods. For novices seeing memory and how it is allocated for these will likely be more memorable than analyzing code for the static keyword.
9. (SCDE) Scope design (e.g., public vs. private fields)

- Fields are class or object variables. public vs. private effects which code can access which fields.
10. (STAM) Static fields and methods
    - Static fields are class variables and static methods are the methods that can access the static fields.
11. (PVR) Primitive vs Reference variables
    - Whether the data type of a variable is one of 8 primitive types or a reference type.
12. (IT2) Understanding loop variable scope
    - Identifying when can the loop variable be accessed.
13. (AR1) Identifying off by one index errors
    - An incorrect access of an array which is a list of variables.
14. (AR2) Reference to array vs array element
    - Arrays are lists of variables. A reference to a list is different than referring to a specific array element.
15. (AR3) Declaring and manipulating arrays
    - Arrays are lists of variables.
16. **(MMR) Memory model, references, pointers**
    - Specifically dealing with memory and accessing memory.
17. **(REC) Recursion, tracing and designing**
    - Recursion is typically implemented by calling the same method multiple times while changing the parameter (variable) values. Recognizing that there are multiple instances of the method, in other words, multiple sets of parameter values at the same time is useful.
18. **(INH) Inheritance**
    - Recognizing that an instance of a class includes allocation of all the instance fields of the class and its ancestors is key to understanding inheritance.

These are important and difficult topics indirectly or not related to data access in

memory:

1. **(PROC) Procedure Design**
   a. Relates to modularization of functionality
2. (BOOL) Construct/evaluate Boolean expressions
   a. Relates to clarity of logic

3. (COND) Writing expressions for conditionals
   a. Relates to clarity of logic in code.
4. (SVS) Syntax vs. Semantics
   a. Differentiates between the symbols used and the meaning of the symbols.
5. (OP) Operator Precedence
   a. Relates to accuracy and readability of the code.
6. (POLY) Polymorphism
   a. Relates to selecting the appropriate method, of several with the same name, based on the values passed to the method.
7. **(APR) Abstraction/Pattern recognition and use**
   a. Relates to reuse of some pattern in the code for solving a problem.
8. (IT1) Tracing nested loop execution correctly
   a. Relates to understanding nesting of loops
9. **(DPS1) Functional decomposition, modularization**
   a. Relates to design and modularization of functionality
10. **(DPS2) Conceptualize problems, design solutions**
    a. Relates to design
11. **(DEH) Debugging, Exception Handling**
    a. Relates to tracking down errors and error handling.
12. (IVI) Interface vs Implementation
    a. Relates to design and external view versus internal view of a class.
13. (IAC) Designing Interfaces, Abstract Classes
    a. Relates to design and external view and partial internal implementation
14. **(DT) Designing Tests**
    a. Relates to testing and verification.

Appendix B: Video Scripts

Anthony Robins (2010) argues that programming language concepts are "unusually tightly integrated". Since we learn on the edge of what we already know, for learners that are unsuccessful early, this results in a negative learning cycle. Our efforts here are to introduce the fundamental programming concepts to novices utilizing a model of computer memory to aid understanding. Our screen casts with narration of our simulator focus on describing very small steps, introducing a few concepts and terms.

Background for Learning Survey 1 script

The key concepts that we are intending the novices to learn in order to learn to read short examples are:

- Declaration: A declaration is just giving a name to an area of memory that we call a variable. A declaration such as "int a;" gives the name "a" to an area of memory.

- Assignment: The '=' sign means put the value from the right hand side into the variable (memory area) on the left side. If a new value is assigned to a variable, the previous value is overwritten and lost.

- Sequence: Program execution is a mechanical process, one step at a time, order of the steps matter.

This script below is identical for both the control and experimental conditions. In fact, the audio narration is the exact same for both conditions, as well. This was achieved by making one recording showing the memory model, experimental condition, and then visually hiding the computer memory portion of the screen cast using a video editing tool to create the control condition. In the experimental condition, computer memory is shown with specific concepts highlighted on the screen. In the control condition computer memory is not shown. In both conditions the length of the screen cast and the narration are identical.

Some specific design aspects of the script are:

- Similar instructions are together. The first time I just explain an instruction. For the second similar instruction, I point out the similarity to a previous instruction, pose a question asking the viewer to guess what the instruction does and then pause for a few seconds. This is to try and engage the learner in actively thinking. Finally, I explain what the instruction does so that the learner can compare to their thinking.

- For the "="sign, which novices probably associate with the mathematical meaning of equality, I both verbally describe the meaning being assignment and not equals as well as show text on the screen with the same points. An arrow is also drawn on the screen, from right-to-left under the assignment statements, to emphasize the asymmetrical, right-to-left processing of the assignment statement.

- For the experimental condition, that shows the computer memory model, I have some short arrows drawn on the screen to highlight aspects of the memory model at the same time the narrator is describing them. I don't highlight every time, in the expectation/hope that the learner will focus on the code and the memory model themselves to develop an understanding of how to read them – essentially learn what we want them to.

- I'm intentionally using the term computer to refer to the computer, the compiler and memory management which is consistent with the insight of Bruce-Lockhart and Norvell (2007). I'm just starting from the terms computer and computer memory that I believe novices will know, particularly since they are utilizing a computer to watch this video.

- I'm intentionally using the term "instruction" throughout, rather than line, statement or expression. Learning to program is essentially learning to provide instructions for the computer. While each line is one instruction in this example, more complex examples may have multiple instructions per line and we would like to avoid needless barriers to future learning. While technically each line in the example is a statement since it ends with ";", since the learner doesn't have to type the semicolon I chose to avoid discussion of semicolon completely right now. While technically statements are composed of expressions I didn't think it would be meaningful to define that term or make the distinction at this time.

- I'm intentionally emphasizing that executing a computer program is a straightforward mechanical process in order to challenge the assumption that novices bring that "…there is a hidden mind somewhere in the programming language that has intelligent interpretive powers." (Pea, 1986)

- While describing declaration statements, I'm intentionally not defining or using the term "declaration" as learning the term is not critical to be able to trace code. This is one way of trying to minimize the content we are asking learners to acquire in this short presentation.

- For integer literals, I just note that the number "10" has value 10. I'm not describing the conversion of a literal from characters to binary as our memory model doesn't show binary. This conversion will need to be described as more details of the computer memory model are defined. Not using the term literal yet either just to simplify the current presentation.

- Just using single character variable names and not defining what an identifier is at this point either, just to keep the presentation simple.

- Only defining the data type "int" as it is actually used in the example. Not discussing any other data types at this point, again just to keep the discussion as simple as possible to focus on teaching how to trace these short examples.

- All the narration is describing the meaning of specific instructions. The meaning or purpose of a set of instructions is never discussed and in fact these example instructions have no purpose other than illustrating the concepts of declaration, assignment and sequence.

Line-by-line script

Step	Focus	Narration

1	```int a;```
	```int b;```
	
	```a = 5;```
	```b = 10;```
	
	```a = b;```
	```b = 7;```
		I'm going to describe the meaning of each instruction in the example computer program shown on the left. The program is written in the Java programming language. I wrote the program by typing in the instructions. Once the instructions are typed in then we ask the computer to execute the program. The computer is a machine that executes the program one instruction at a time.

| 2 | `int a;` | <step>The computer executes the program instructions, one at a time, in order. We start with the first instruction "int a;". <step>"int a" means give an area of computer memory the name 'a'. <step>More specifically, "int" is short for integer and means that memory location "a" will be used to store an integer. An integer is a whole number, such as 1, 2, 3, 99, 100 etc.   The memory location 'a' will only contain a single number at any time. The memory location 'a' is also referred to as variable 'a' since the value at that memory location can change, as we will see. |
| 3 | `int b;` | <step>Now the computer executes the next instruction of the program "int b".<step>

Note that this instruction is very similar to the previous instruction. Can you guess what this instruction does? <pause><step> As you may have guessed, this instruction gives the name 'b' to another area of computer memory. In other words, it creates the variable 'b'. Note that 'b' will also be used to hold an integer which is a whole number, a |

		number without a decimal.
4	`a = 5;`	<step>The next instruction is an assignment statement. <step> The instruction is read "a" is assigned the value "5". Note that I did not say 'equals'. I said "a" is assigned the value "5". In a Java program the equals sign is referred to as the assignment operator. The assignment operator means take the value on the right side and put it into the memory location named on the left side. Note that order is important; the right side is executed and then the result is stored in the memory location named on the left side of the assignment operator.
5	5	<step>'5' refers to the number 5. Note that variables such as 'a' and 'b' refer to memory areas, while numbers such as '5' refer to the integer value 5. When the computer executes this program, it puts the number 5 into memory.

6	`a`	<step>As we described earlier the name 'a' refers to a memory location that will store one number.
7	`a = 5`	<step>Assignment means to copy the value from the memory location where '5' is to the memory location named 'a'. Note that this is a copy, the '5' remains in the original location. Note also that if memory location 'a' had a value, it is now overwritten. The only value in memory location a is now '5'. Since some program instructions effect the contents of memory, such as this one, when tracing through a program, as we are doing, it is important to keep track of the current contents of memory.
8	`b = 10;`	<step>The next instruction in our program is similar to the previous. This instruction is read "b is assigned the value 10". Can you guess what this instruction does? <pause> <step> This instruction tells the computer to put <step>the value 10 into <step>the memory location named b.<step> <pause>

9	`a = b;`	<step>Now the computer executes the next instruction in our program. Note that this instruction is similar to the previous instruction but instead of a number on the right side there is the variable 'b'. What do you think happens in this case? <pause> <step> In this instruction, <step>the value in memory location 'b' is copied/assigned to <step>the memory location 'a'. Remember that assignment means to take the value from the <step>right hand side and put it into the memory location on the left hand side. <pause>
10	`b = 7;`	<step>Finally the computer executes the last instruction in our program. What do you think may happen to the current value that is in 'b'?<pause><step>This instruction says assign the value 7 to the memory location "b". Note that whatever value is in 'b' currently will be overwritten as the variable 'b' can only hold one value. The number 7 from our program is <step>put into a memory location by the computer. When this assignment instruction is executed, the value 7 is copied to <step>the memory

		location 'b' <step>overwriting the value that was previously there.
11		<step> Thank you for taking the time to watch this video.

The following videos are available via YouTube. They are unlisted and therefore require the following links to access.

Instruction without computer memory:

https://www.youtube.com/watch?v=FIhThMCrdpI

Instruction with computer memory: https://www.youtube.com/watch?v=ObZ7mA5EkuI

Experiment 2 Video Scripts

We created an example that includes 2 instructions demonstrating variable-to-variable

assignment as that seemed to be a stumbling point by many in experiment 1. Also, some

prior results suggested that maybe our video covered "too much, too quickly" so we

shortened to 3 minutes. To cut the time, we abbreviated our discussion of declaration

and assignment of initial values and the discussion related to literals. At the beginning

of the video we added a statement describing the purpose of the example. At the end

we mentioned the purpose again.

Step	Focus	Narration

1	`int c;`
	`int d;`
	`int e;`
	
	`c = 5;`
	`d = 10;`
	
	`c = d;`
	`e = c;`
		This short video describes the first steps in learning how to program a computer. The purpose of these instructions is to show how to put a number into an area of memory and then copy the number from one area of memory to another.

2	`int c;`	<step 5> We start with the first instruction "int c;". "int c" tells the computer to give an area of computer memory the name 'c'. initially the value within "c" is undefined. "int" is short for the word integer which is a number, such as 1, 2,

		99, 100 etc.

3	`int d;`
	
	`int e;`
		Can you guess what the next instruction does? \<pause>
		
		\<step>This instruction gives the name 'd' to another area of memory. \<step>And you can probably guess that 'int e' will give the name 'e' to a third area of memory.

4	`c = 5;`	\<step>In the next instruction, when you see the symbol that looks like the equals sign, imagine an arrow pointing to the left. We call this symbol assignment. Even though it looks like an equals sign it does Not mean equals. This instruction means to put the number on the right side of the assignment symbol into the memory area named 'c' on the left side. So, the value 5 is put into the memory area 'c'. If another number was in memory area 'c', that number is now gone and is replaced with the new number, in this case 5.
5	`d = 10;`	\<step>What do you think "d is assigned 10" means? It is very similar to the previous instruction. If you said, puts the number 10 into memory area 'd', you are right!

| 6 | `c = d;` | <step>Remember that when you see the assignment symbol, that looks like the equals sign, think of an arrow pointing left. So, this instruction means copy the number that is in memory area 'd' and put it into memory area 'c'. The number in area 'd' is not changed, just the area 'c' is changed since it is on the left side of the assignment symbol. Note that 'c' no longer contains the number 5 but 'c' now contains the number 10 that was just copied there. |
| 7 | `e = c;` | This next line is tricky so be careful. Do you think 'c' contains the number 5 that was put into 'c' originally? Or does 'c' now contain the number 10? <pause> If you said 'c' contains the number 10 and the number 10 will be put into memory area 'e', you are correct! The *last* value in 'c' is the value that will be copied to 'e'.

As you can see the values in 'c', 'd' and 'e' vary so we call 'c', 'd' and 'e' variables.

To summarize, the purpose of these lines of Java instructions is simply to demonstrate creating variables and copying |

		values between them.

We added a third group to our experiment, a no instruction group. The script below

was recorded as the no instruction group video.

Step	Focus	Narration

1	`int c;`
	`int d;`
	`int e;`
	
	`c = 5;`
	`d = 10;`
	
	`c = d;`
	`e = c;`
		This video shows some instructions for programming a computer. The instructions shown are written in the Java programming language. The purpose of these instructions is to demonstrate some fundamental programming concepts.

		Part of our research is to understand what knowledge beginner's bring that may help them make sense of programming instructions. Following this video are some programming examples that look similar to the one shown here. Without having someone explain the precise meaning of the instructions, please make an attempt to answer the following questions. At the end, there is an opportunity to comment if you have insight you would like to offer us.

YouTube links for the videos. These videos are unlisted meaning that one must have the link in order to view them.

Instruction without computer memory: http://youtu.be/76RfvVVixsg

Instruction with computer memory: http://youtu.be/P4ZEcolWgk4

No Instruction video: http://youtu.be/ufAVwIsX4DE

Appendix C: Complete Survey

Learning Survey 1

The following are screen shots of the actual survey were taken in preview mode on the

UWM Qualtrics survey site. Note at the bottom of each screen shot is only a next page

button (>>) for the participant to navigate through the survey.

University of Wisconsin – Milwaukee
Consent to Participate in Online Survey Research

Study Title: Learning Survey

Person Responsible for Research: (Primary Investigator) Susan McRoy, (Student Primary Investigator) James Williams

Study Description: The purpose of this research study is to gather information about how people learn from video. Approximately 250 subjects will participate in this study. If you agree to participate, you will be asked to complete an online survey, including watching a 5 minute video, that will take approximately 10 minutes to complete. The questions will ask for demographic information, previous experience with the content and content questions related to the video.

Risks / Benefits: Risks to participants are considered minimal. Collection of data and survey responses using the Internet involves the same risks that a person would encounter in everyday use of the Internet, such as breach of confidentiality. While the researchers have taken every reasonable step to protect your confidentiality, there is always the possibility of interception or hacking of the data by third parties that is not under the control of the research team.

There will be no costs for participating. If the Amazon Mechanical Turk website was your means of arriving at this survey than any benefits are as described on the Amazon Mechanical Turk website.

Limits to Confidentiality: Identifying information such as your name, email address, and the Internet Protocol (IP) address of this computer will not be asked or available to the researchers. Data will be retained on the Qualtrics website server for 1 year and will be deleted by the research staff after this time. However, data may exist on backups or server logs beyond the time frame of this research project. Data transferred from the survey site will be saved on a password protected computer for 1 year. Only Susan McRoy and James Williams will have access to the data collected by this study. However, the Institutional Review Board at UW-Milwaukee or appropriate federal agencies like the Office for Human Research Protections may review this study's records.

Voluntary Participation: Your participation in this study is voluntary. You may choose to not answer any of the questions or withdraw from this study at any time without penalty. Your decision will not change any present or future relationship with the University of Wisconsin Milwaukee.

Who do I contact for questions about the study: For more information about the study or study procedures, contact Susan McRoy at mcroy@cs.uwm.edu or (414) 229-6695, James Williams at willi329@uwm.edu or (608-438-6790).

Who do I contact for questions about my rights or complaints towards my treatment as a research subject? Contact the UWM IRB at 414-229-3173 or irbinfo@uwm.edu

Research Subject's Consent to Participate in Research:
By entering this survey, you are indicating that you have read the consent form, you are age 18 or older and that you voluntarily agree to participate in this research study.

Thank you!

I have read, understood, and printed a copy of, the above consent form and desire of my own free will to participate in this study.

○ Yes

○ No

>>

What is your gender?

◯ Female

◯ Male

What is your age?

[   ]

What is the highest level of education you have completed?

◯ Some High School

◯ High School Graduate

◯ Some College, no degree

◯ Associates degree

◯ Bachelors Degree

◯ Graduate Degree (Master's, Doctorate, etc.)

Where are you located?

◯ Africa   ◯ Canada

◯ Antarctica   ◯ Mexico

◯ Asia   ◯ Central America

◯ Oceania (Australia, New Zealand, etc.)   ◯ South America

◯ Europe   ◯ Middle East

◯ USA   ◯ West Indies
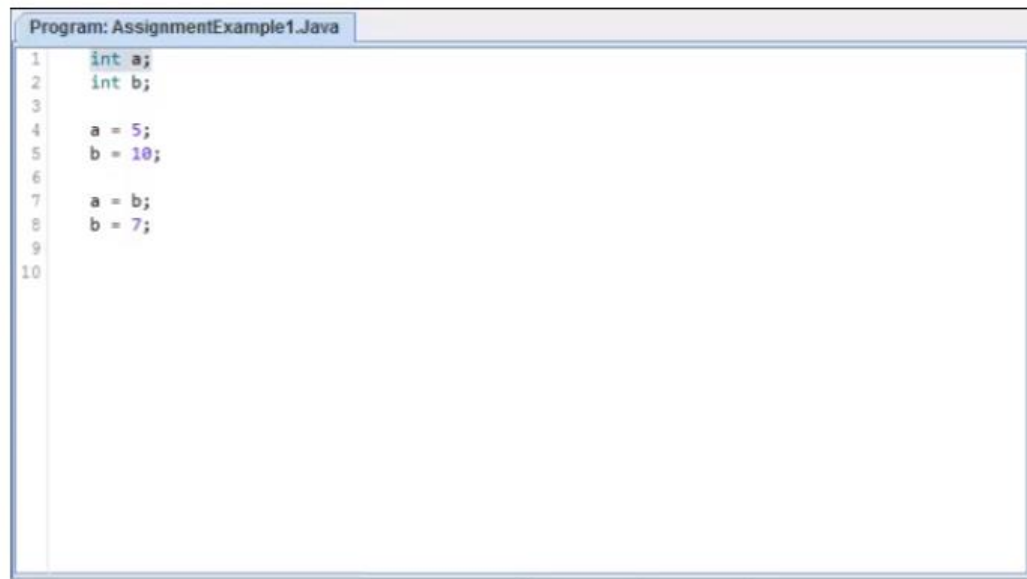
[>>]

Have you ever had training on how to write a computer program either in a classroom or online?

○ Yes

○ No

>>

This 5 minute video is intended for beginning programming students to understand basic computer programming instructions. After watching the video, please answer the following questions.

Program: AssignmentExample1.Java

```
1      int a;
2      int b;
3
4      a = 5;
5      b = 10;
6
7      a = b;
8      b = 7;
9
10
```

>>

Have you ever had a fatal heart attack while working at your computer?

- ○ Yes
- ○ No

`>>`

In the boxes provided below, write the values in the variables after the following code has been executed:

```
int r;
int s;

r = 2;
s = 4;
r = s;
```

The value in r is: [   ]

and the value in s is: [   ]

In the boxes provided below, write the values in the variables after the following code has been executed:

```
int p;
int q;

p = 1;
q = 8;
q = p;
```

The value in p is: [   ]

and the value in q is: [   ]

In the boxes provided below, write the values in the variables after the following code has been executed:

```
int x;
int y;
int z;

x = 5;
y = 3;
z = 7;

x = z;
y = x;
z = y;
```

The value in x is: [   ]

and the value in y is: [   ]

and the value in z is: [   ]

How confident are you that you answered the last 3 questions correctly?

	Not at all				Completely confident
	1	2	3	4	5
Confidence	○	○	○	○	○

[>>]

In one sentence that you should write in the box below, describe the purpose of the following three lines of code for any set of values stored in variables i, j, k:

```
j = i;
i = k;
k = j;
```

[   ]

[>>]

The purpose of the following three lines of code is to swap the values in variables a and b:

```
c = a;
a = b;
b = c;
```
In one sentence that you should write in the box below, describe the purpose of the following three lines of code, for any set of possible initial integer values stored in those variables:

```
j = i;
i = k;
k = j;
```

[   ]

How confident are you that you have answered the last 2 questions correctly?

	Not at all				Completely confident
	1	2	3	4	5
Confidence	○	○	○	○	○

[>>]

Please enter any comments about this learning survey in the box below.

[   ]

[>>]

Thank you for participating in our research project!

Enter the code: **UWMAMT1**
on the Amazon Mechanical Turk site to get credit for taking this survey.

Learning Survey 2 and 3

The survey questions were identical between Learning Survey 2 and 3. The videos

shown were the only differences.

University of Wisconsin – Milwaukee
Consent to Participate in Online Survey Research

Study Title: Learning Survey

Person Responsible for Research: (Primary Investigator) Susan McRoy, (Student Primary Investigator) James Williams

Study Description: The purpose of this research study is to gather information about how people learn from video. Approximately 500 subjects will participate in this study. If you agree to participate, you will be asked to complete an online survey, including watching a short video, that will take approximately 10 minutes to complete. The questions will ask for demographic information, previous experience with the content and content questions related to the video.

Risks / Benefits: Risks to participants are considered minimal. Collection of data and survey responses using the Internet involves the same risks that a person would encounter in everyday use of the Internet, such as breach of confidentiality. While the researchers have taken every reasonable step to protect your confidentiality, there is always the possibility of interception or hacking of the data by third parties that is not under the control of the research team.

There will be no costs for participating. If the Amazon Mechanical Turk website was your means of arriving at this survey than any benefits are as described on the Amazon Mechanical Turk website.

Limits to Confidentiality: Identifying information such as your name, email address, and the Internet Protocol (IP) address of this computer will not be asked or available to the researchers. Data will be retained on the Qualtrics website server for 1 year and will be deleted by the research staff after this time. However, data may exist on backups or server logs beyond the time frame of this research project. Data transferred from the survey site will be saved on a password protected computer for 1 year. Only Susan McRoy and James Williams will have access to the data collected by this study. However, the Institutional Review Board at UW-Milwaukee or appropriate federal agencies like the Office for Human Research Protections may review this study's records.

Voluntary Participation: Your participation in this study is voluntary. You may choose to not answer any of the questions or withdraw from this study at any time without penalty. Your decision will not change any present or future relationship with the University of Wisconsin Milwaukee.

Who do I contact for questions about the study: For more information about the study or study procedures, contact Susan McRoy at mcroy@cs.uwm.edu or (414) 229-6695, James Williams at willi329@uwm.edu or (608-438-6790).

Who do I contact for questions about my rights or complaints towards my treatment as a research subject? Contact the UWM IRB at 414-229-3173 or irbinfo@uwm.edu

Research Subject's Consent to Participate in Research:
By entering this survey, you are indicating that you have read the consent form, you are age 18 or older and that you voluntarily agree to participate in this research study.

Thank you!

I have read, understood, and printed a copy of, the above consent form and desire of my own free will to participate in this study.

○ Yes

○ No

```
>>
```

What is your gender?
- ○ Female
- ○ Male

What is your age?

[   ]

What is the highest level of education you have completed?
- ○ Some High School
- ○ High School Graduate
- ○ Some College, no degree
- ○ Associates degree
- ○ Bachelors Degree
- ○ Graduate Degree (Master's, Doctorate, etc.)

Where are you located?
- ○ Africa
- ○ Antarctica
- ○ Asia
- ○ Oceania (Australia, New Zealand, etc.)
- ○ Europe
- ○ USA
- ○ Canada
- ○ Mexico
- ○ Central America
- ○ South America
- ○ Middle East
- ○ West Indies

[>>]

Have you ever had training on how to write a computer program either in a classroom or online?
- ○ Yes
- ○ No

[>>]

This video is intended for beginning programming students to understand basic computer programming instructions. After watching the video, please answer the following questions.

```
das7 m: AssignmentExample2.Java
1       int c;
2       int d;
3       int e;
4
5       c = 5;
6       d = 10;
7
8       c = d;
9       e = c;
10
```
0:00 / 3:01

Timing

These page timer metrics will not be displayed to the recipient.

First Click: *5.11 seconds.*

Last Click: *5.11 seconds.*

Page Submit: *0 seconds.*

Click Count: *1 clicks.*

>>

Have you ever had a fatal heart attack while working at your computer?

○ Yes

○ No

>>

In the boxes provided below, write the values in the variables after the following code has been executed:

```
int r;
int s;

r = 2;
s = 4;
r = s;
```

The value in r is:

and the value in s is:

How confident are you that you answered the last question correctly?

	Not at all				Completely confident
	1	2	3	4	5
Confidence	○	○	○	○	○

>>

In the boxes provided below, write the values in the variables after the following code has been executed:

```
int p;
int q;

p = 1;
q = 8;

p = q;
q = p;
```

The value in p is:

and the value in q is:

How confident are you that you answered the last question correctly?

	Not at all				Completely confident
	1	2	3	4	5
Confidence	○	○	○	○	○

>>

In the boxes provided below, write the values in the variables after the following code has been executed:

```
int x;
int y;
int z;

x = 5;
y = 3;
z = 7;

x = z;
y = x;
z = y;
```

The value in x is:

and the value in y is:

and the value in z is:

How confident are you that you answered the last question correctly?

	Not at all				Completely confident
	1	2	3	4	5
Confidence	○	○	○	○	○

>>

The purpose of the following three lines of code is to swap the values in variables a and b:

```
c = a;
a = b;
b = c;
```

In one sentence that you should write in the box below, describe the purpose of the following three lines of code, for any set of possible initial integer values stored in those variables:

```
j = i;
i = k;
k = j;
```

[   ]

How confident are you that you have answered the last question correctly?

	Not at all				Completely confident
	1	2	3	4	5
Confidence	○	○	○	○	○

>>

In one sentence that you should write in the box below, describe the purpose of the following three lines of code for any set of values stored in variables i, j, k:

```
j = i;
i = k;
k = j;
```

[   ]

How confident are you that you have answered the last question correctly?

	Not at all				Completely confident
	1	2	3	4	5
Confidence	○	○	○	○	○

>>

Please enter any comments about this learning survey in the box below.

>>

Thank you for participating in our research project!

Enter the code: **UWMAMT3**
on the Amazon Mechanical Turk site to get credit for taking this survey.

James Stephen Williams
Curriculum Vitae

EDUCATION

Ph.D.	**University of Wisconsin - Milwaukee**	2014

Computer Science, minor in Educational Psychology
Thesis: *A Computer Learning Environment for Novice Java Programmers that Supports Cognitive Load Reducing Adaptations and Dynamic Visualizations of Computer Memory*
Tool: ReadJava program simulator

M.S.	**University of Colorado at Colorado Springs**	1997

Computer Science, emphasis in Artificial Intelligence
Thesis: *Natural Language Processing and the Minimalist Program*

B.S.	**University of Colorado at Colorado Springs**	1992

Computer Science

AWARDS & HONORS

"Teacher of the Year" award for the Herzing University system, 2005
"Outstanding Graduate Student" award in Department of Computer Science, University of Colorado at Colorado Springs, 1997

TEACHING AND ACADEMIC EXPERIENCE

Western Governors University, Salt Lake City, Utah, 7/10 - 1/12
Learning Resources Specialist – Information Technology
- Identified and acquired 21st Century learning resources for one of the nation's leading online competency-based institutions.
- Negotiated with vendors and customized content to ensure WGU students received a top-of-the-line experience from their learning resources.

Herzing University, Madison, Wisconsin, 5/04 - 12/07
Associate Professor (April 2007 to December 2007)
Assistant Professor (May 2004 to March 2007)

- Taught over 40 courses in Computer Science and Networking including Java, Visual Basic, C++, Programming Logic, Computer Networks, Computer Applications, TCP/IP, Business Systems Analysis, Web Site Development, Web Programming – Open Source, and Windows Scripting.
- Presenter at "Taste of Technology" for high school juniors and seniors
- Presenter at "Reach and Teach" - free technology training for local non-profits
- Presented on "Active/Collaborative Learning" for both Full-time and Adjunct faculty
- Organized all 9 members of an advanced C++ course into a Project Team to develop and deliver BlackJack game using the Agile methodology.

University of Phoenix, Milwaukee Campus, Wisconsin, 9/02 - 6/04
Instructor

- Taught courses in various information systems technologies, including Computers and Information Processing, Introduction to Operating Systems, and Computer Programming using Java.

Batky-Howell, Inc., Englewood, Colorado, 11/99 - 7/02
Instructor

- Taught 86 short-courses including Introductory Unix/Linux, Shell Scripting, Advanced Unix Tools, Linux Programming, Java Programming, Advanced Java Programming, JavaServer Pages, Rational Rose and Object-Oriented Analysis and Design with the Unified Modeling Language.
- Students were software development professionals at primarily Fortune 500 companies.
- Reviewed and revised technical training materials.

ACADEMIC SERVICE

Assessment Committee, Chair, Herzing University, Madison campus, 2005-2007
Computer Science Department, Chair, Herzing University, Madison campus, 2006-2007

Computer Science Sub-Committee of College Curriculum Committee, Chair,
Herzing University, 2006-2007
Led review and updating of the Computer Information Systems curriculum,
Herzing University, 2005

GRANTS

Innovation Grant for Interdisciplinary Robot Project, $620, 2005
Grant for Parallax SumoBot Robot kits and associated materials.
Innovation Grant for Programming Robots, $675, 2006
Grant for Lego Mindstorms Robot kits and associated materials.

PROFESSIONAL ASSOCIATIONS

Member of Association of Computing Machinery (ACM)
Member of Computer Science Teachers Association (CSTA)

PUBLICATIONS

Williams, James S., and Jugal K. Kalita. "Parsing and Interpretation in the Minimalist Paradigm", Computational Intelligence, Volume 16, Number 3, August 2000.

Williams, James S., "Natural Language Processing and the Minimalist Program," Master's Thesis, University of Colorado at Colorado Springs, March 1997.

Bergacker, Dave, Jim Williams and Jugal Kalita, "Issues in Planning Realistic Motion from Natural Language Instruction," AAAI Workshop on Spatial and Temporal Reasoning, Seattle Washington, August 1994.

Williams, Jim and Jugal Kalita, "Understanding Instructions for Physical Action," Third Golden West International Conference on Intelligent Systems, Las Vegas Nevada, June 1994.

INDUSTRY EXPERIENCE

Pronto Spanish Services, LLC, Lake Mills, Wisconsin, 8/02 - present
Co-founder

- As co-founder, guided the strategic direction, business and contract negotiations, and all technical efforts for an occupational Spanish company.
- Led the design and development of a custom online registration and learning management system (LMS), using PHP, MySQL, HTML, CSS, JavaScript, and JQuery on a Linux hosting platform in order to provide unique capability in our niche industry.
- Developed technical integration between our online hosting platform and the learning management systems of key providers.

TargetSmart, Inc., Denver, Colorado, 7/96 - 2/99
Lead Software Engineer
- Performed Object-Oriented Design and implementation of the database interface and conversion subsystem for the TargetSmart and ProspectSmart database marketing products.
- Designed and implemented object hierarchies, an interpreted language and various parsing engines. Implemented in Delphi.

MCI, Colorado Springs, Colorado, 1/94 - 11/95
Programmer/Analyst
- Performed structured analysis, design and C implementation of a call record Match/Merge system on a Sun/Solaris platform using Sybase.
- Collaborated to diagnose and resolve in the field networking, database and switch record problems.

Infotec Development Inc., Colorado Springs, Colorado, 5/91 - 7/92
Programmer
- Converted Computer Based Training lessons from Regency code to TenCORE on Space Shuttle systems.
- Programmed Computer Based Training lessons in CAST on HP/Motif workstations.

Rocky Mountain Skyline Bookstore Association, Colorado Springs, 5/90 - 8/90
Software Engineer
- Gathered requirements, designed and implemented MS-DOS database software in Modula-2 to merge textbook lists and print out final cross-referenced copy.
- Installed software and trained users.

CERTIFICATIONS
CompTIA Network+ Certified Professional
Sun Certified Programmer for the Java 2 Platform