EXPLORING COMPUTER SCIENCE WITH Lynx

Written by: Thomas Walsh Jr. PhD, Teacher / Author

Right to Use Lynx

To get full benefit from this e-book, the user should get a Lynx Annual Subscription either an Individual Plan or a School / Club Plan. There is a time-limited Free Trial for Lynx but this would require the User to cram all the lessons in a short time frame. This Free Trial also limits the number of projects a user can save.

Before deciding what plan to choose, if any, a user can simply click on the Create a Lynx Project (red button on the home page of lynxcoding.club). This will allow the user to explore Lynx but saving is not possible since no user account will be registered.

Important note for Canadians: The Government of Canada has purchased Subscriptions for Canadian residents therefore they have nothing to pay.

© LCSI excluding those sections indicated as © Thomas Walsh Jr., 2020. All rights reserved. The document contained herein may not be reproduced or photocopied, stored in retrieval systems or transmitted, in any form or by any means, electronic, mechanical, recording or otherwise, without the prior approval of Logo Computer Systems Inc. Notwithstanding the foregoing, schools, libraries and individuals that have a valid license to use the Lynx coding app found at https://lynxcoding.club may copy and distribute, for use within the school, library or personal residence, all pages except pages 6 – 8 written by Seymour Papert.

Lynx is a trademark and LCSI is a registered trademark of Logo Computer Systems Inc.

© 2020 Thomas Walsh Jr., Registered United States Copyright Office, Library of Congress

Table of Contents

Preface **5**

Section 1: Introduction
What is Logo? And Who Needs It? *by Professor Seymour Papert* 6
Microworlds, Computational Thinking, and 21st Century Learning 9

Section 2: Using Lynx to Introduce Computer Science **17**
Acknowledgements 17
Teacher and Parent Reflections 18
List of Figures 19
Logo Coding for Essential Skills, Cognitive Development, and Learning Benefits Using
Teacher Mediated Scaffolding 20
Teacher Lesson Plans **29**
Lesson 1: Introduction to Lynx Procedures and Turtle Commands 29
Lesson 2: The Repeat Command and Geometric Shapes 30
Lesson 3: Introducing Turtle Programs 31
Lesson 4: Creating Modular and Recursive Programs 32
Lesson 5: Assigning Variable in Logo Programming 33
Lesson 6: Animating Turtle Shapes with a Slider and Adding Features 34
Lesson 7: Going Further: Words and Lists in Logo Procedures 35
Lesson 8: Applying Graphics, Animation, and Interactive List Procedures for
Developing Games 36
Teacher Activities Answer Guide and Resources **37**
Turtle Degrees Answer Key 37
Lynx Turtle Shapes Answer Guide 38
Lynx Observation Form (Lessons 1-3) 39
Lynx Rubric Evaluation (Lessons 4-8) 40
Turtle Primitives Flashcard Cut Outs* 41

Section 3: Introducing Lynx in Eight Lessons **51**
Navigating the Lynx Platform **51**
Layout Windows Design 51
User Guide Support: Getting Started and List of Primitives 52
Lynx User Policy 53
Saving Projects and Sharing with Friends 53
Drawing Turtle Graphics **54**
Additional Primitive Drawing Commands 54
Lynx Program Project – Turtle Commands 56
Changing Pensize, Graphic Color, and the Fill Command **56**
Lynx Program Project – Colors 58
Repeat It! **58**
What Does Repeat Do? 58
Lynx Program Project - Repeat 59

Table of Contents (Continued)

Introducing Turtle Programs **60**
Getting Inside the Turtle's Backpack to Run Programs on a Click 61
Adding a Button to Your Lynx Project 62
Lynx Program Project – Procedures 63
Creating Modular Programs **64**
Lynx Program Project – Modular Procedures 67
Simple Logo Recursion **68**
Lynx Program Project – Modular Recursive Procedures 70
Assigning Variables in Logo Programming **70**
Lynx Variable Program Project 76
Recursive Variable Modular Procedures in Logo Programming **76**
Lynx Modular Variable (Recursion) Project 78
Animating Turtle Shapes **79**
Adding a Button and Slider for Animation of Shapes 80
Animation Procedures with Varying Shape Speed and Added Background 83
Additional Feature for Project Development **84**
Adding Pages 84
Adding Sound and Music 84
A Clickable and Detectable Turtle to Control Movement 84
Lynx Animation Program Project 85
Going Futher: Words and Lists in Logo Procedures **85**
The Print Statement and Character String Changes 85
Defining and Manipulating Words, Lists, and Sentences 87
Words and Lists in Logo Program Procedures 92
Lynx Words and Lists Program Project 94
Interactive Lists and Numbers Programs **94**
Lynx Interactive Lists and Numbers Program Project 97
Applying Graphics, Animation, and Interactive List Procedures for Developing Games **97**
Interactive Game Project 98
Appendix: Resources & Activities **99**
Turtle Primitives 100
Turtle Degrees 101
Lynx Turtle Shapes 103
Repeat Predictions 104
A-Mazing 106
Cognitive Monitoring Planning 108
Cognitive Monitoring Student Project Example 111
Changing Procedures and Predicting Skills 113
Multiple Turtles 115
A Turtle Calculator Application 116
Turtle Degree Clock 118
Logo Program Procedures Learning Models from Figures 119
Guided References and Resources **131**

Preface

This book was written to support teachers wishing to introduce coding to their students, ages 9 to 14. This curriculum provides a teaching methodology that introduces basic turtle commands and procedures found in the Logo programming language. The lessons progress introducing graphics programming, coding features (i.e., variables and recursion), animation, use of words and lists, and developing gaming projects. The extent of teacher material coverage, and the suggested lesson time period, will depend on the grade level and capabilities of the students. Turtle activity ideas provided at the end of each lesson suggests differentiated learning opportunities for students.

To become familiar with the educational philosophy of Dr. Seymour Papert, and Logo itself, it is suggested that the teacher read the **Section 1** introduction.

Teachers will gain additional insight by reading the article in **Section 2**: *Logo Coding for Essential Skills, Cognitive Development, and Learning Benefits Using Teacher Mediated Scaffolding* describes teaching elementary and middle school students using coaching and teacher scaffolding techniques; along with support for student cognitive benefits in learning Logo based on implementing more carefully planned teacher-directed lessons using teacher-mediated instruction.

Section 3 explains, for students and teachers, Logo commands and procedures with examples of student figures related to the topic areas. The teacher may prefer to photocopy the lesson plans as handouts and/or provide this e-book for student on-screen reading and/or use a Smart Board or LCD projector to display pages for all students to see. Student may find it helpful to have a copy of the e-book on their computer desktop to copy program codes to the Procedures Tab and access the URL links provided. The Turtle Hints, throughout the text, provide further guidance in using the Lynx program (e.g., coding tips and use of tools).

The **Appendix Resources** and **Activities** are integrated into the lessons to support instructional learning and may be provided as handouts. The Turtle Degree Clock with turning activities and A-Mazing has been found to assist students in learning degrees to support development of Logo graphics. Other activities have been found to assist students learning Logo primitives, developing understanding of the `repeat` command, and using words and lists number procedures. Flash cards are included for the most frequently-used commands and may be printed for display on a bulletin board for easy reference. Cognitive monitoring along with predicting procedure outcomes support student problem solving development and program planning skills. The coding procedure examples illustrated in the figures are available in the appendix for student viewing and ideas for developing their projects. Suggested use of the Lynx *observation form and rubric evaluation* is provided for the lessons, and may be helpful as a tool for mini-conferencing with students to focus development of their individual Logo projects.

Section 1: Introduction

What is Logo? And Who Needs It? ***by Professor Seymour Papert***
Extracted from ***Logo Philosophy*** http://www.microworlds.com/company/philosophy.pdf
(**Note**: If a link does not work inside the pdf then copy and paste the URL into your browser.)

What is Logo?

I have myself sometimes slipped into using an answer given by many Logoists in the form of a definition: "Logo is a programming language plus a philosophy of education" and this latter is most often categorized as "constructivism" or "discovery learning." But while the Logo spirit is certainly consistent with constructivism, there is more to it than any traditional meaning of constructivism and indeed more to it than "education." The right answer to "what is Logo" cannot be "An X plus a Y." It is something more holistic and the only kind of entity that has the right kind of integrity is a culture and the only way to get to know a culture is by delving into its multiple corners.

Logoists reject School's preoccupation with getting right or wrong answers. What others might describe as "going wrong" Logoists treat as an opportunity to gain better understanding of what one is trying to do. Of course rejecting "right" vs. "wrong" does not mean that "anything goes." Discipline means commitment to the principle that once you start a project you sweat and slave to get it to work and only give up as a very last resort. Life is not about "knowing the right answer" – or at least it should not be – it is about getting things to work!

The frame of mind behind the Logo culture's attitude to "getting it to happen" is much more than an "educational" or "pedagogic" principle. It is better described as reflecting a "philosophy of life" than a "philosophy of education." But insofar as it can be seen as an aspect of education, it is about something far more specific than constructivism in the usual sense of the word. The principle of getting things done, of making things — and of making them work – is important enough, and different enough from any prevalent ideas about education, that it really needs another name. To cover it and a number of related principles (some of which will be mentioned below) I have adapted the word constructionism to refer to everything that has to do with making things and especially to do with learning by making, an idea that includes but goes far beyond the idea of learning by doing.

I want to emphasize here what might for educational decision-makers be the most important difference between the "n word" constructionism and the "v word" constructivism. The v-word refers to a theory about how math and science and everything else is learned and a proposal about how they should be taught. The n-word also refers to a general principle of learning and teaching, but it also includes a specific content area that was neglected in traditional schools but which is becoming a crucial knowledge area in the modern world.

Choosing constructivism as a basis for teaching traditional subjects is a matter for professional educators to decide. I personally think that the evidence is very strongly in favor of it, but many teachers think otherwise and I respect their views. But the constructionist content area is a different matter. This is not a decision about pedagogic theory but a decision about what citizens

of the future need to know. In the past most people left the world only slightly different from how it was when they found it. The rapid and accelerating change that marks our times means that every individual will see bigger changes every few years than previous generations saw in a lifetime. So this is the choice we must make for ourselves, for our children, for our countries and for our planet: acquire the skills needed to participate with understanding in the construction of what is new OR be resigned to a life of dependency.

A crucial aspect of the Logo spirit is fostering situations that the teacher has never seen before and so has to join the students as an authentic co-learner. This is the common constructivist practice of setting up situations in which students are expected to make their own discoveries, but where what they "discover" is something that the teacher already knows and either pretends not to know or exercises self-restraint in not sharing with the students. Neither deception nor restraint is necessary when teacher and student are faced with a real problem that arises naturally in the course of a project. The problem challenges both. Both can give their all.

I like to emphasize this last point by the following analogy. The best way to become a good carpenter is by participating with a good carpenter in the act of carpentering. By analogy the way to become a good learner is by participating with a good learner in an act of learning. In other words, the student should encounter the teacher-as-learner and share the act of learning. But in school this seldom happens since the teacher already knows what is being taught and so cannot authentically be learning. What I see as an essential part of the Logo experience is this relationship of apprenticeship in learning. Logo, both in the sense of its computer system and of its culture of activities, has been shaped by striving for richness in giving rise to new and unexpected situations that will challenge teachers as much as students. In so doing, the Logo culture approaches teachers as intellectual agents.

It is important to recognize – only slightly simplifying a complex issue—two wings of digital technology: technology as an informational medium and technology as a constructional medium in which garb it is more like wood and bricks and steel than like printing or television. Of course the two wings are equally important; but popular perception is dominated by the informational wing because that is what people see and ceaselessly hear about and that is what reflects the predominant role of informational media in their lives.

This one-sidedness in perception of technology has produced a deep distortion of how people think about its contribution to education. This has happened because education itself has two wings that also could be called "informational" and "constructional." Part of learning is getting information that might come from reading a book or listening to a teacher or by visiting sites on the Web. But that is only one part of education. The other part is about doing things, making things, constructing things. However here too there is an imbalance: in large part because of the absence of suitable technologies, the constructional side of learning has lagged in schools, taking a poor second place to the dominant informational side.

Before making my final point let me review some of the features of the Logo culture that I have mentioned in relation to the chapters of this book.

The Logo programming language is far from all there is to it and in principle we could imagine using a different language, but programming itself is a key element of this culture.

So is the assumption that children can program at very young ages.

The assumption that children can program implies something much larger: in this culture we believe (correction: we know) that children of all ages and from all social backgrounds can do much more than they are believed capable of doing. Just give them the tools and the opportunity.

Opportunity means more than just “access” to computers. It means an intellectual culture in which individual projects are encouraged and contact with powerful ideas is facilitated.
Doing that means teachers have a harder job. But we believe that it is a far more interesting and creative job and we have confidence that most teachers will prefer “creative” to “easy.”

For teachers to do this job they need the opportunity to learn. This requires time and intellectual support. Just as we have confidence that children can do more than people expect from them we have equal confidence in teachers.

We believe in a constructivist approach to learning. But more than that, we have an elaborated constructionist approach not only to learning but to life. We believe that there is such a thing as becoming a good learner and therefore that teachers should do a lot of learning in the presence of the children and in collaboration with them.

We believe in making learning worthwhile for use now and not only for banking to use later. This requires a lot of hard work (we’ve been at it for thirty years) to develop a rich collection of projects in which the interests of the individual child can meet the powerful ideas needed to prepare for a life in the twenty- first century.

My belief is that the Logo philosophy was not invented at all, but is the expression of the liberation of learning from the artificial constraints of pre-digital knowledge technologies.

MicroWorlds, Computational Thinking, and 21st Century Learning

"*Understanding procedures and processes is important in math. There's a fantastic way to do that – it's called programming."(*Conrad Wolfram: *Teaching kids real math with computers*, http://www.ted.com/talks/conrad_wolfram_teaching_kids_real_math_with_computers.html)

In these section, references are made to MicroWorlds EX, MicroWorlds or MW. This was an older version of Logo designed around 2005. The newest version of Logo is Lynx, designed in 2019 and 2020.

Human/computer Mutualism

As we have changed technology, technology has also changed us – especially in how we think about thinking and seek new ways to solve the many questions and problems we face. Technology has radically enhanced communication and global collaboration and made it easier to carry out vast numbers of complex, yet routine calculations. It has, also, at a different level, and maybe even more importantly, provided us with a medium in which to develop new patterns of thinking. As scientists, whether in the area of physical, health, or social sciences, are influenced by computer science, they have gained new perspectives on how to approach both problems, old and new, and innovation in research design and interpretation.

Computational Thinking – An Essential Skill for the 21st Century

Computers have freed us from the onerous and sometimes impossible task of running long, complex calculations, the type often required in research, so that researchers now can more easily focus on the big ideas and patterns that emerge. In thinking as a computer scientist, researchers become aware of behaviors and reactions that can be captured in algorithms or can be analyzed within an algorithmic framework.

This way of thinking - computational thinking - now gives them a different framework for visualizing and analyzing, a whole new perspective. To rephrase a common idiom, "Until you have a screwdriver, everything looks like a nail."

Computational thinking depends on a variety of skills (logic, creativity, algorithmic thinking, modeling/simulations), involves the use of scientific methodologies, and helps develop both inventiveness and innovative thinking. It has roots in mathematics, engineering, technology, and science, and, in the synthesis of ideas from all these fields, has created a way of thinking that is only just beginning to generate enormous changes and benefits.

Thinking about Thinking through Programming

Just using computers does not necessarily lead to the development of computational thinking. Facebook, Twitter, Flickr, Google, while all great applications, do not require or involve the same skills. Computational thinking is a learned approach and there's no better way to learn it than through programming. Programming employs all the components of computational thinking and the knowledge gained through the experience of tackling programming challenges – both explicit and tacit - can provide a cognitive framework not only for computer science, but for any field, from natural and health sciences to the social sciences and humanities.

So, here we have an important, essential and very truly 21st century "skill"- computational thinking - that is best learned through experience, interactions, actively doing. It allows students who learn to express themselves through programming (and who have the time to gain this knowledge) to not only answer questions but also generate new ones as they begin to view these challenges through the lens of the tacit knowledge intrinsic to computational thinking.

A student, when using programming to tackle a question, has to develop a hypothesis as to how best to solve or answer it, then build, through analysis of the problem, a set of rules (an algorithm) that can be used to test the hypothesis, after which she can review the results (data), and revise the solution. The art of programming requires creativity and inventiveness, logic, algorithmic thinking, and an appreciation of the recursive nature of this process, as the student learns from her failures, refines her work, and gets a deeper understanding of the problem. As with any creation, even once a solution is found – a pattern, an algorithm – the solution can be refined, simplified and beautified, made more elegant. In a way, programming provides the same satisfaction as a video game – the opportunity to find a path – one of many - through a problem. (It's logical – video games are created by programmers!) The difference here is that students can answer their own questions and create their own challenges.

Patterns and Algorithmic Thinking

With applications such as MicroWorlds EX children have the opportunity to develop their computational thinking abilities. The approach to thinking that exploring with MicroWorlds helps develop can become a lens for how they understand and frame ideas and tackle challenges in all areas of the curriculum.

Using MicroWorlds EX, students experiment with mathematical ideas such as number, angles and geometric shapes, variables, and recursion. As they gain experience by experimenting with turtle commands, students can begin to sequence instructions and see the outcomes, hypothesizing as to which sequence creates the result they want and then testing their ideas. It is through this sequence of actions – seeing a pattern, creating a rule (an algorithm) that describes that pattern and then testing to see if the logic is correct – repeated over time and in a playful, exploratory approach – that learners begin to develop a new perspective on how to approach questions/challenges in other areas. This is particularly powerful if a teacher is there to coach them as they think about what they did to tackle a previous challenge and how that can be used to tackle the next in other to help them think about their learning.

As anyone who has used a computer during the last few decades realizes, there's no limit to what computer programmers can create – and MicroWorlds EX opens this door to all learners, no matter what their age. As students explore different MicroWorlds commands, they begin to understand how a series of actions can create a specific result. Run an instruction, see a reaction. It's the core of data collection (and its useful to highlight this link).

Next, combine some instructions and see the result. Combine them in a variety of ways to see how the results each time may differ. This extends the exploration.

Once students are introduced to the idea that by grouping several instructions in a specific order they not only get a desired result, but an object that may be useful in multiple situations.

They can preserve this algorithm by creating a procedure, a user-defined, localized command to do a specific task. In this way, students begin to extend the MW language in a very personalized way. A project can now include newly created programmed objects that can be used in other instances or copied into other projects, shared with other people, or manipulated in different ways.

Turtle graphics is a classic starting point for exploring patterns with MicroWorlds, as students use the turtle as an object with which to explore. One doesn't have to look beyond what is often a first exploration with Logo – drawing a square.

A student may readily understand that to draw a square using the MicroWorlds turtle, one should type something similar to this:
Instruction set #1:

`forward 100`	Moves the turtle forward a specified number of pixels or 'turtle-steps'
`right 90`	Pivots the turtle a specified number of degrees to the right
`forward 100`	
`right 90`	
`forward 100`	
`right 90`	
`forward 100`	

In this set of instructions, the turtle moves forward 100 pixels to create each side of the square and pivots 90 to create each corner. At this point, the turtle has returned to its starting position on the screen, but in order to be pointing in the same direction as when it began, the student would need to add:

```
right 90
```

A clear pattern emerges - each set of two instructions (a forward and a right instruction) are repeated four times. At this point, the student may chose to use the repeat command. This command repeats a list of instructions a specified number of times:
Instruction set #2:

```
repeat 4[forward 100 right 90]
```

Either set of instructions draws a square, the desired outcome.

According to www.mathworld.wolfram.com, an algorithm is "a specific set of instructions for carrying out a procedure or solving a problem." The set of instructions to draw a square are a springboard to looking for patterns in order to create other geometric shapes. How would one create a similar representation of a pattern in order to draw a triangle? What makes a triangle a triangle and not a different shape? What kind of triangle can you draw?

What if the following instruction is typed by mistake?

```
repeat 4[forward 90 right 100]
```

Instead of this being an error, this is an opportunity to analyze what happened. The student may ask: Why did this happen? How many times do I need to repeat these instructions to get back

to the place where I started? The shape begins to look like a star. Could it be that stars have similar patterns? What shape is this? (A torus) How can I make a larger/smaller space in the center? In other words, the student learns how to ask questions, how to look for patterns, how to test ideas, and how to create instructions (rules) that let her either repeat the same pattern or adapt it.

For beginning programmers, having a good coach–one who helps them look for patterns and figure out from the data (the drawing) why the resulting drawing occurred-is critical. Although the student may be able to recognize the patterns and understand the data, a good coach can help focus on meta-cognitive skills by asking well-designed, guiding questions.

Recursion

Another aspect of computational thinking is an understanding and recognition of recursive patterns. Patterns are often repeated in mathematical solutions, nature, in other areas of science, and often these patterns are repeated with modifications – modifications that can be codified since they are based on some rule. Recursion is the process of describing an action in terms of itself. (This will become clearer below.) The more one plays around with the idea of recursion, the more one begins to recognize its presence in other disciplines.

The following procedure shows a simple example of recursion through programming. (Procedures extend the MicroWorlds EX vocabulary with words that you define yourself. A procedure is a group of instructions with a name that you assign to it. When you define a procedure, the name becomes part of the MicroWorlds EX vocabulary for that project.)

```
to move
forward 1
move
end
```

The word `to` and the procedure name begins every procedure definition. This is the first instruction. It tells the turtle to move forward one pixel.

The second instruction says run this procedure again. All procedures end with the word `end` – letting MicroWorlds know this is the end of the procedure.

Each time the `move` procedure runs, the turtle moves forward one pixel (one “turtle step”) and then runs `move` (again), which tells MicroWorlds to move the turtle forward one pixel and then run move (again), and so on. Notice that the built-in command “forward” requires an input to tell it how much the turtle should move forward.

This is recursion at its most basic level. By changing the procedure slightly, various effects can be observed.

```
to move :step
forward :step move :step + 1
end
```

`:step` is a variable, standing in for a value. Now the turtle moves forward whatever value is provided for `:step` The next time move is run, `:step` will increase by 1.

The `move` procedure now requires an input, just as `forward` requires an input. Now, to run the `move` procedure, an initial value for `:step` (which stands for 'the variable named step') needs to be added, so the instruction would be:

```
move 1
```

Now each time the `move` procedure runs, the turtle moves forward whatever the value of :step is. The first time, the turtle moves forward one pixel and then runs `move` (again), but now the value of `:step` increases by one. This time when the `move` procedure runs, MicroWorlds EX moves the turtle forward *two* pixels and then runs `move` (again) increasing `:step` by 1 again, and so on. In this version of move, the turtle doesn't just continue to move forward, but it accelerates (eventually moving so fast that it becomes a mere blur on the screen).

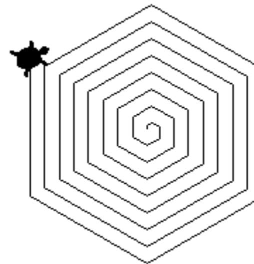
Being able to express the rules of a recursive pattern through programming helps students better understand and recognize these patterns. Here is another example:

```
to spiral :step :angle
if :step = 100[stop]
forward :step
right :angle
spiral :step + 2 :angle
end
```

`if :step = 100[stop]` — This conditional statement says if `:step` equals 100, the procedure stops. This prevents the turtle from spiraling forever.

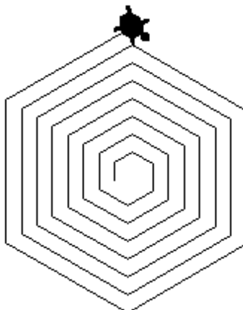
`right :angle` — Right tells the turtle to pivot a specific number of degrees.

`spiral 2 60` would look like this when run:

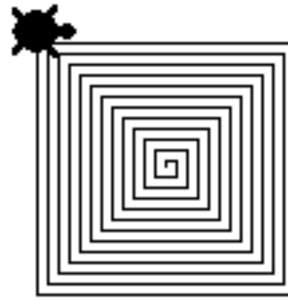


Programming provides an opportunity to play with pattern rules to see the effect of simple changes. For example:

`Spiral 4 60` — the first side is 4 pixels long, but the spiral is very similar to the previous one.

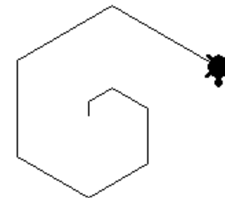


`Spiral 2 90` - the angle is different..



Changing the procedure also creates different results:

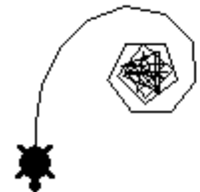
```
to spiral :step :angle
if :step = 100 [stop]
forward :step right :angle
spiral :step + 10 :angle
end
```



```
to spiral :step :angle
if :angle = 360 [stop]
forward :step right :angle
spiral :step :angle + 5
end
```

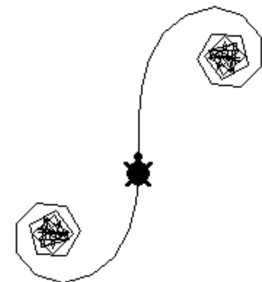
Notice the change here.

....and here

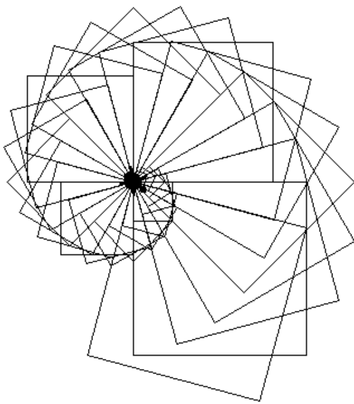


```
to spiral :step :angle
if :angle = 720 [stop]
forward :step right :angle
spiral :step :angle + 5
end
```

….and here



And with some more exploration..



Compare these two images:

Through an understanding and recognition of these patterns students will gain both experience and tacit knowledge of this form of patterning that may provide new ways to understand our world from the smallest forms of matter to the largest.

Transfer of Learning

The learning explorations made possible through the type of programming described above and the resulting development of computational thinking is of value to the learner in the immediate context, but if these skills and ways of interacting with the world transfer to other domains, the impact would be greatly amplified, leading to new patterns of thinking in different knowledge domains and an innovative, inventive perspective on finding new solutions to old problems.

Do these computational thinking skills transfer and positively impact learning in other domains? Anecdotal evidence of this transfer seems to indicate it does.

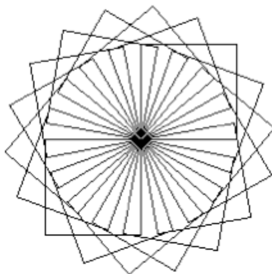


Figure 1. A subprocedure for creating a square is shown within a superprocedure for creating a flower. Procedures in MicroWorlds Ex can become part of larger procedures. The included ones are called subprocedures and the enclosing programs are called superprocedures.

```
to square
repeat 4[forward 50 right 90]
end
to flower
repeat 18[square right 20]
end
```

(Peter Skillen, *The Construction Zone*)

Peter Skillen, Manager, former Social Media Professional Development with the YMCA of Greater Toronto, provides the following story to provide evidence of transfer in a blog post entitled *Deep Understanding & the Issue of Transfer* (http://theconstructionzone.wordpress.com/2010/03/07/deep-understanding-the-issue-of- transfer/):

Jeffrey, a Grade 2 student, made a most interesting leap from Logo to a completely different

domain one day.

We were having a discussion inspired by the flight of the space shuttle piggybacked on a jumbo jet. Our Grades 2/3 class had the opportunity to watch the flight. When we returned to the classroom, a discussion of space naturally arose. One child asked if Earth was in space, and in asking the question, she determined it must be, because it wasn't sitting on anything. The discussion continued until Jeffrey piped up.

"You know . . . it's sort of like Logo." We stopped and looked at him curiously. "What do you mean?" I asked him studiously.

He replied, "Well, Earth is like a procedure. It's like a subprocedure inside the solar system. The solar system is the superprocedure. And the solar system is like a subprocedure in- side the universe. The universe is like the superprocedure."

"Fascinating," I said, then asked, "What's the biggest superprocedure?"

After a moment he replied, "I don't know. I guess the universe."

Peter continues, "I was truly amazed at the generalization across domains that Jeffrey had made. He clearly demonstrated significant transfer of a concept from his experiences with Logo to an authentic event. Although Jeffrey's illumination happened spontaneously, I learned that I could play an important role in helping students to acquire [Gavriel] Salomon's 'effects of' [technology] by providing opportunities for them to look for these comparisons across subject areas."

Opportunities to Explore

There are few opportunities in most school curricula to explore recursive patterns (although recursive patterns appear throughout nature and mathematics), develop and test algorithms, invent solutions to student-generated questions, or see their world through a different lens. The teacher/coach plays a key role in helping learners reflect on their thinking in order to bring about these lasting changes in metacognition. And, programming with products such as MicroWorlds EX and MicroWorlds JR create opportunities for even young children to explore big ideas as they develop true 21st century thinking skills.

"Computational thinking will be a fundamental skill used by everyone in the world. To reading, writing, and arithmetic, lets add computational thinking to every child's analytical ability." Jeannette Wing, Professor of Computer Science and Department Head, Computer Science Department, Carnegie Mellon University http://www.youtube.com/watch?v=C2Pq4N-iE4I

"The role of the teacher is to create the conditions for invention rather than provide ready-made knowledge." Seymour Papert, professor emeritus, MIT/MIT Media Lab.

Section 2: Using Lynx to Introduce Computer Science

For Students Aged 9 to 14

By Thomas Walsh Jr. PhD
Teacher and Author

Acknowledgments

The project is a result of the guidance, direction, and encouragement provided by faculty and major professor Dr. Ann Thompson at Iowa State University. Her inspiration to learn Logo resulted in a dissertation topic on “The Implementation and Evaluation of a Sequential, Structured Approach for Teaching LogoWriter to Classroom Teachers.” She also provided editorial expertise to support publishing two journal articles from the dissertation on a literature review and Logo staff development. Additional support is credited to Dr. Mi Ok C. Lee, graduate student colleague, for her collaboration and use of the effective cognitive monitoring strategy for students.

Further encouragement and direction is credited to teachers and administration in the Ames Community Schools supporting implementation of the Logo curriculum during my teaching career. A special recognition is given to third and sixth grade students who have provided insight into their learning of the Logo language and development of progamming skills. Credit is awarded to students and parents who provided permission for use of their projects for motiviating and encouraging peers in learning programming skills. In addition, recognition is given to the Ames Middle School math teachers for their engagement with students in learning Logo and their contributing quotations for the text.

Learning of more advanced coding procedures and use of tools is credited to teaching short courses to upper elementary and middle school high ability students for the Early Outreach Program at Iowa State University. My additional learning of coding skills along with use of tool applications, especially in developing game projects, was developed as teacher scaffolding and team instruction was provided to the students.

A sincere thanks is given to Michael Quinn and Susan Einhorn, LCSI publishers, who have provided the encouragement, editorial expertise, and leadership to promote this valuable technology curriculum for students. Michael Quinn’s collaborative support in revision of the manuscript has been instrumental and invaluable for improving the e-book evolving from use with MicroworldsEX to the Lynx coding platform.

It is the hope the e-book made available as a free downloadable publication will not only provide support in developing student coding skills, through teacher scaffolding with mediated instruction, but encourage enthusiasm in the learning of computer programming.

Teacher and Parent Reflections:

Microworlds is a rich, flexible learning environment. All of the strengths of the LOGO programming language are included (and enhanced!). The original intent of Seymour Papert to provide a Piagetian learning environment is well preserved here. The multiple window programming environment makes experimentation easy and enjoyable. Even early-age students can easily master important important programming ideas like command syntax, debugging, subroutines/functions, variables, conditionals, and recursion. More importantly, problem solving skills, mathematical reasoning, and analytic skills are all strengthened in a really fun, engaging way. Kids love working with the program and sharing their ideas in a group setting!

Philip Wagner
Parent of two MicroWorlds EX users, aged 10 and 12
New York City

MicroWorlds (Logo) is equally engaging for both boys and girls. It is a valuable extension of geometric concepts. It provides a nice progression from basic geometry to advanced concepts.

Stacey Culhane
Grade 6 Math Teacher

MicroWorlds (Logo) provides interaction of the kids with the technology and an introduction to basic programming. It provides a concrete understanding of geometric figures primarily with angles, length of lines, and perimeter. I see great value in the opportunity to be creative and experiment with the technology.

Jeremie Knutson
Grade 6 Math Teacher

Provides students an opportunity to explore their ability to use technology in a positive way. Forces them to think outside the box and be creative. Kids cruise through the program applications with much success.

Craig Sengbusch
Grade 6 Resource Teacher

List of Figures*

Figure 1. Lynx layout features available from the Ecosystem activity and also available from other cards on the Lynx web site.
Figure 2. Window display example for `repeat` primitive when hovering over command.
Figure 3. A student gear project using setpensize, setpos, and fill commands.
Figure 4. Student swim goggle created with turtle primitives and repeat instructions.
Figure 5. Student smileyface program created with turtle primitives and repeat instructions.
Figure 6. Student arrow program using turtle commands with repeat instructions and added buttons.
Figure 7. Lynx student project showing a computer laptop program with comments explaining coding procedures.
Figure 8. A student modular clock program with circle and number subprocedures.
Figure 9. A student modular program with smiley, snake, and peace subprocedures.
Figure 10. A student modular neighborhood program with house, tree, window, circle, and square subprocedures.
Figure 11. A student modular broken key piano program with various position piano part subprocedures.
Figure 12. A student modular recursive named TX2 superprocedure calling g4 subprocedure.
Figure 13. A student modular recursive superprocedure, including random color changing command, calling shape subprocedures.
Figure 14. A student modular variable program procedures creating different shoe colors.
Figure 15. A student solar eclipse variable program with subprocedures.
Figure 16. A student modular outfit procedures with assigned variable shirt values.
Figure 17. A student modular crayons procedures with assigned variable values.
Figure 18. A student modular variable desk program using assigned variable values.
Figure 19. A student variable project clearing and creating medieval colored swords with buttons.
Figure 20. View of the Rundog program entered in the Lynx program.
Figure 21. An example of a Lynx race animation using a control buttom to adjust the speed of one Lynx.
Figure 22. View of window for creating a button for the Lynx race.
Figure 23. View of the tool window showing the slider name speed with minimum and maximum values.
Figure 24. Example of two Lynx racing at various speeds with the addition of a background shape.
Figure 25. Demonstration of the Meet program presenting a question in the Lynx program.
Figure 26. Demonstration of the Meet program displaying the final greeting in the Lynx program text box.
Figure 27. A student interactive words and lists project called Dudetalk.
Figure 28. Help User Guide link of project ideas
Figure 29. A green turtle pursuit to reach the four black turtles through a maze.

*The author was granted permission to display student examples used in the book.

Logo Coding for Essential Skills, Cognitive Development, and Learning Benefits Using Teacher Mediated Scaffolding

Thomas Walsh Jr.

Introduction

Learning computer languages while in school or study in computer science gives students the skills needed to learn new computer languges easily (Bureau of Labor Statistics, 2020). One computer language experience is coding in the Logo language. Empirical and meta-analysis research studies support of teaching Logo programming in developing student cognitive problem-solving skills has been documented. Using guided instruction with teacher-mediated scaffolding has been found as an effective method in preparing students using the Logo code programming language to create geometric graphic, animation, and gaming projects. Anecdotal benefits in teaching coding are presented along with research on Logo's contribution to student learning. Research on potential benefits, using teacher-mediated or guided instruction is discussed along with curriculum methodology for teacher delivery of Logo coding skills to students balancing teacher direction with planned discovery. Teacher scaffolding strategies are presented including cognitive monitoring. Anecodotal student outcome benefits in learning Logo coding along with differential instruction are discussed. More research will be needed on specific teacher mediation intervention techniques to facilitate successful transfer of problem-solving skills from Logo to other domains including coding in other languages.

Current Demand for Programmers and Coders

The teaching of coding has been gaining support based on media publications and advocates from industry and nonprofits reporting the need for computer science programmers. Labor market trends support the long-term demand for computer programmers who have knowledge of a variety of programming language experiences. Computer literacy and knowledge of IT skills has become regarded as an essential skill for students in the 21st century to develop problem solving skills.

Support for teaching computer science or "computational thinking" and bringing coding to the classroom has been reported in the press (Stross 2012, Naughton 2012 & Schmidt 2012). Former software engineer and co-founder of the Holberton School of Software Engineering Sylvain Kalache reports coding is important because it's all around us:

> *From the smartphone in our pocket, to the smart watch on our wrist, it's also launching rockets in space or controlling our fridge," says Kalache. All industries are disrupted by software and even if not all of us will become Software Engineers, all of us will be interacting with it, so it's important to understand the foundations of it." (Stenger, 2017).*

Computer-oriented jobs are the number one source of all new wages in the United States and are in demand four times more than any other occupation according to Cameron Wilson, COO and VP of Government Relations for Code.org (Wills, 2016). Wills reports computer science majors are the number-one major hired by volume and the second highest post-undergraduate earners. The Bureau of Labor Statistics (2020) reports employment of computer and information technology is projected to grow 12 percent from 2018-2028, much faster than the average for all occupations, with a seven percent decline in domestic (U.S.) computer programmers due to hiring offshore. This decline may be due to the fact coding can be done anywhere with many

programmers working from their homes. The Bureau of Labor Statistics also reports programmers who have knowledge of a variety of programming languages and keep up to date with the newest coding tools will improve job prospects. Pisani (2018) reports there is currently a shortage of computer engineers, and teaching students to code will ensure a pipeline of future talent to hire.

Computer programming or coding it's now regarded as an essential ability for 21st century learners and is becoming a key component of many curriculums including instruction at the primary school level. Singer (2017) reports Code.org, an industry-backed nonprofit, goal is to get every public school in the United States to teach computer science. Code.org has helped to persuade two-dozen states to change their education policies and laws while creating free introductory coding lessons. Amazon has announced the Amazon Future Engineer program to pay for summer camps, teacher training, and other initiatives to benefit kids and young adults form low-income families to learning coding to spur students to study computer science (Pisani 2018). Pisani reports Microsoft and Facebook, have also committed cash to bring coding to schools, which could ultimately benefit the companies.

Knowledge of basic IT skills will be a literacy requirement given the growth of technology. Vlatko (2015) reports with this technology growth more countries are introducing programming as part of their syllabus including European countries and Canada or introducing a digital curriculum as part of the STEM initiative in Australia and Singapore. The CanCode Program is investing $60 million over two years from 2019 to 2021 to support coding and digital skill development to Canadian youth (K-12) and provide teachers with professional development (CanCode, 2018). A 2016 Gallup report found that 40% of American schools now offer coding classes compared to just 25% a few years ago (Stenger, 2017).

Anecdotal benefits of learning coding to develop critical thinking, on task persistence or determination, problem-solving through debugging, processing skills, trial and error (Heggart 2014, Porter 2016 & William 2017) along with improved social skills and self-confidence (An 2017 & Morris 2017) has been documented.

Logo's Potential Benefits

Given these trends students will need to be prepared to learn programming code language, preferably starting at an earlier age. Research on Logo's contribution to student learning has appeared in the literature during the last three decades. Seymour Papert and his colleagues at the Massachusetts Institute of Technology (MIT) Artificial Intelligence Laboratory developed Logo in the late 1960's. Papert believes that Logo has no threshold and no ceiling, meaning that the programming instruction can be used for applications with young children to secondary student, across the curriculum.

Research on potential benefits, using teacher-mediated or guided instruction, is summarized as follows (Walsh, 1994):

- Contributing to understanding of geometric concepts
- Facilitating students' understanding of geometric conceptualizations and thinking (e.g., understanding of angle sizes and geometric shapes)
- Increasing understanding of geometric transformation (i.e., symmetry, slides, and rotations)

- Supporting the development of cognitive and metacognitive skills (e.g., planning skills) including measures of creativity
- Improving problem solving in decomposition skills, error recognition, and feedback
- Gains in divergent thinking, field dependence/independence (i.e., relationship of figures), and impulsivity/reflectivity.

Some empirical research examining problem-solving using Logo have shown no significant benefits or positive cognitive effects derived from Logo instruction. These studies have reported limited if any effects in students' ability in learning to solve equations, solution sets, mapping, conditional logic, geometry, and planning skills. A study by Littlefield et. al. (1989) report methodological considerations (clearly defining training conditions, documenting programming mastery, and transfer measures) have not been considered to evaluate the claims that learning Logo can enhance children's general thinking skills. These authors also report the importance of considering the method of teaching Logo and its effects on the development and transfer of general thinking skills from the Logo environment to non-Logo problems. This includes consideration of:

- The teacher's approach.
- When, where and how often the teacher intervenes.
- Whether attempts are made to relate the Logo programming to other problem situations.
- The number of students who share a computer at the same time.
- The nature of the student interactions (Littlefield et. al. 1989, 335-336).

The study of student learning providing structured and unstructured learning environments found support for goal-oriented structure in the training program using mediated teacher intervention. Littlefield et. al. report the features of mediation that apply directly to Logo instruction include framing, which involves the act of relating specific sets of behaviors to a broader framework of problem-solving (for example, breaking down Logo subprocedures into manageable components). Another feature is bridging, which involves the act of relating processes that occur within one context to similar processes occurring elsewhere (for example, using mediation to relate right and left turn degrees to time on a clock).

Further support for teacher mediation and scaffolding is provided in two meta-analyses conducted by Alfieri et. al. (2011) using a sample of 164 studies examining the effects of discovery learning practices. Most of these studies involved teaching domains in math, science, problem-solving, and computer skills. The results of the first meta-analysis indicate that unassisted discovery does not benefit learning. The analysis also found direct teaching is better than unassisted discovery, provide learners with worked examples, and use of timely feedback. The implications here suggest students benefit when provided with examples of Logo programs and procedures as learning models. The study also reports that students may benefit from individualized feedback on homework assignments with worked examples provided.

The second meta-analyses suggest that teaching practices should employ scaffolding tasks that require learners to explain their own ideas. These authors report that feedback, worked examples, scaffolding, and elicited explanations are needed for learners to be redirected, to some extent, when they are mis-constructing. Research supporting these student benefits are based on providing more structured presentation of coding procedures with programming model examples

accompanied by teacher feedback, mediation and scaffolding of student learning (Littlefield et. al. 1989 & Alfriei et. al. 2011). Alfieri et. al. elaborate on this idea stating:

> *Feedback, scaffolding, and elicited explanations do so in more obvious ways through an interaction with the instructor, but worked examples help lead learners through problem sets in their entireties and perhaps help to promote accurate constructions as a result* (Alfriei et. al. 2011, 12).

The findings suggest that unassisted discovery does not benefit learners, whereas feedback, worked examples, scaffolding, and elicited explanations do.

Support in the research has suggested that Logo experiences using teacher-mediated instructional practices produce positive near transfer (e.g., debugging Logo programs transfers to map reading directions) and far transfer (i.e., to higher order thinking in another content subject area). Other potential benefits of students working cooperatively in pairs have included more time on-task in problem solving, correcting of program errors, and benefits in self-esteem social skills.

Microworlds Curriculum Using Teacher Scaffolding

To prepare students for the future workplace and with research support for learning Logo coding a curriculum program is needed. *Exploring Computer Science with MicroworldsEX* (Walsh, 2013-2017) e-book was developed as a structured learning methodology of learning activity lessons, with opportunities for discovery and exploration, to support student learning in a "Microworlds" project-based environment to create geometric graphics, animation, and gaming using the Logo programming language. The curriculum was developed from the author's 30-year Logo teaching experience with elementary and middle school regular and gifted education students, along with dissertation research and journal publications (Walsh 1992-1993 & 1994) supporting use of guided instruction for student learning programming code. Guided instruction was found for the potential cognitive benefits for teaching Logo to be achieved by implementing more carefully planned teacher-directed lessons balanced with student problem solving and discovery learning using teacher-mediated scaffolding. In 2020 the e-book was revised for use with the Lynx Logo platform titled *Exploring Computer Science with Lynx.*

Working with students to develop their programming skills requires curriculum support with handout information about turtle primitives, along with examples of programming procedures. The teacher provides the scaffolding and guided questions to support student development of workable program procedures. Students can approach programming using a top down strategy (in other words, writing code directly into programs and testing outcomes in the Command Center) or bottom up strategy (students test parts of the program in the Command Center and paste pieces of workable code into program procedures). Teachers will find themselves learning with the students as they discover innovative ways to use and apply program procedures. Since teacher time is usually limited, students should learn to debug procedures, for instance, by testing code line by line in the Command Center or working with student teams to solve their problems.

Cognitive Monitoring

The e-book includes a cognitive monitoring strategy has been used with grade three and grade six students in developing a student guided programming project. The strategy (Lee, 1990) involves having students draw the desired Logo graphic outcome by hand, decomposing the steps

to write a program, writing a plan, writing codes or subprocedures, testing and identifying errors, and debugging the program. For example, a student wants to draw a house. The decomposed shapes identified are a triangle on top of a square, and this is hand-drawn as the planned graphic. The student writes the plan as a program with a roof (the triangle) and a square. The executed program may be written as follows:

```
to house
repeat 3 [fd 100 rt 90]
fd 100 rt 90 fd 100 rt 90
rt 45 lt 45
end
```

The executed graphic created with this program turns out not to be a house, which means the student must debug the program and keep trying the new versions until the desired drawn graphic outcome is achieved.

Student Reported Outcomes

The author has found numerous benefits for teaching Logo to students representing regular classroom education third and sixth graders as well as gifted sixth graders enrolled in an Extended Learning Program (ELP). These students have reported benefits from learning the Logo language to develop coding projects. These reports have been substantiated by teacher observation, staff comments, and anecdotally collected information from students. Problem-solving skills noted have included:

- Using guessing and checking by breaking the program into smaller parts
- Thinking in steps and order by repeating use of some commands over and over again
- Experimenting and testing procedures in the Command Center and then writing these into a program
- Experimenting with angles and variable commands to make programs work
- Making predictions, testing commands, and trying different programs
- Using previous knowledge and examples to begin a project
- Creating a mental picture and having the turtle start in one place.

Student expressed benefits in learning Logo programming in support of math skills has been found to include learning mathematical operations (for example, addition, subtraction, multiplication, and division), measuring distance, working with angles and degrees, learning about geometry and shapes, understanding coordinate graphing skills (for example, using the `setpos` command), and developing programming procedure knowledge (for example, variable and recursion). Other computer skills students have reported learning include keyboarding (typing skills); copy, cut, and pasting; and learning how the computer works. Many students have stated that Logo programming has improved their thinking skills, mental work, problem solving, and planning skills. Some students have expressed that one has to be accurate, careful, and follow directions when programming.

These benefits in Logo coding have been substanciated by high ability students in grades 5/6 and 7/8 enrolled in the Early Outreach Program (EOP), formerly OPPTAG, a short course of study at Iowa State University. Many of these students developed coding animation projects and games. Some of these projects have included PS4 game controller, a button coding survey quiz,

pong game, coin flip program, moving chess board, rubric cube program, trivia game, and animated dragon program. This information about the benefit of Logo coding was expressed orally or reported on the class summative evaluation by the students.

Student Differentiation

Teaching Logo coding with elementary and middle school students over the years reaffirms that a broad student population with diverse levels of achievement, ages, gender, and cultural diversity can learn a programming language. Students of different nationalities, non-English language learners (ELL), and students identified with various exceptionalities (for example, TAG, Integrated Services for Behavior, and a plan for students with disabilities) all develop the skills necessary to successfully produce Logo coding projects. Logo coding experiences, using program platforms like Lynx, provides "built in" differentiation for instruction in which the completed projects can vary in complexity depending on student aptitude and interest. Both males and females have been successful in this learning environment.

Use of cognitive monitoring and instruction in debugging skills supports a diverse student population. The cognitive monitoring strategy involves student planning skills, metacognitive thinking, and problem solving. Some students realize their problem is too difficult or easy to solve, and they then need to evaluate their initial graphic goal. Showing students how to debug procedures, including pasting programs in the Command Center and pressing return after each instruction line, is a helpful strategy for finding and fixing code errors. One high ability third grade student developed an elaborate and lengthy coding procedures, then asked for help to find the program error. The debugging strategy was particularly helpful for this student since his error was on a procedure past 100 lines of coding and requiring lengthy instuctor time given 24 other students in the class needed assistance with "hands in the air." This third grader ended up enrolling in math classes at the university at the end of his elementary year and entered as a student in mathematics the following year.

Whlle Logo has been identified as a programming experience for students transiting from block or picture coding prior to coding in more complex languages like Python and JavaScript (Logo Computer Systems, Inc. 2020), for some students Lynx may be their first programming experience. This may be especially true for younger elementary students. Given this reality students may need instruction in some prepatory or concurent learning prior to coding with Logo. Teaching turtle degree turns and request a visual turtle clock model while working on coding projects may be needed. A turtle commander games provides students practice with right and left degree turns. For example, in this activity the instructor stands in back of the room asking students to stand and gives them one or a series of commands to turn a number of degrees (for example, right 90 left 45 or `rt 90 lt 45 rt 120`). Students can also be directed to move forward and backward a number of turtle steps (using one 'foot space' increments), for example, forward 5 (`fd 5`). Other support activities have included practice telling time using degrees along with flash cards of frequently used coding commands. Collaborative teaching with math and resource teachers is helpful when providing additional support for some students needing differentiated instruction. The activities in the Lynx e-book curriculum provide activities in teaching degrees and learning coding commands (flash cards) with tiered differentitated project ideas to accommodate diverse students learners.

Conclusion

The need to provide coding experiences to students has been discussed to promote "computational thinking" and problem solving skills, which may support students in pursuing computer science careers. Logo was discussed as a coding language with the potential to achieve cognitive benefits. These benefits are given when more carefully planned teacher-directed lessons are balanced with student problem solving and planned discovery using teacher-mediated scaffolding. Support for more carefully planned, teacher-directed lessons during initial introduction and learning of Logo skills is provided in the literature. *Exploring Computer Science with Lynx* provides a curriculum methodology for teacher delivery of Logo coding skills to students balancing teacher direction with planned discovery. Teachers will also need to serve as facilitators to provide student support by scaffolding student questioning and directing independent Logo programming exploration. More research will be needed on specific teacher mediation intervention techniques, in addition to better understanding what is required to facilitate successful transfer of problem-solving skills from Logo to other domains including coding in different languages.

The enduring impact of the curriculum has been evident when encountering former students and the first question they ask is: *Do you still teach with the turtle?*

References

Alfieri, L., Brooks, P. J., and Aldrich, N. J. and Tenenbaum, H. R. (2011). "Does Discovery-Based Instruction Enhance Learning?" *Journal of Educational Psychology* 103 (1): 1-18.

An, E. (2017). "7 Benefits You'll Notice When You Start Learning to Code." *CareerFoundry*. Retrieved from: https://careerfoundry.com/en/blog/web-development/7-benefits-of-learning-to-code/

Bureau of Labor Statistics. (2020). U.S. Department of Labor, *Occupational Outlook Handbook*, Computer Programmers. Retrieved from: https://www.bls.gov/ooh/computer-and-information-technology/computer-programmers.htm

CanCode. (2018). Government of Canada. From: Innovation Science and Economic Development Canada. Retrieved from: https://www.ic.gc.ca/eic/site/121.nsf/eng/home

Heggart, K. (2014). "Coded for Success: The Benefits of Learning to Program." *Edutopia*. Retrieved from: https://www.edutopia.org/discussion/coded-success-benefits-learning-program

Lee, M. (1990). *Effects of guided Logo programming instruction on the development of cognitive monitoring strategies among college students*. Unpublished PhD dissertation, Iowa State University.

Littlefield, J. Delclos, V. R., Bransford, J. D., Clayton, K. N. and Franks, J. J. (1989). *Cognition and Instruction* 6 (4): 331-366.

Logo Computer Systems Inc. (2020). Lynx [Computer software]. Retrieved from: https://lynxcoding.club/

Morris, S. (2017). "8 Ways Learning to Code Can Benefit You Right Now." *Skillcrush*. Retrieved from: https://skillcrush.com/2017/01/30/learn-to-code-benefits/

Naughton, J. (2012). "Why All Our Kids Should Be Taught How to Code." *The Guardian*. Retrieved from: http://www.guardian.co.uk/education/2012/mar/31/why-kids-should-be-taught-code

Pisani, J. (2018). Amazon's new goal: Teach 10 million kids a year to code. AP News. Retrieved from: https://www.apnews.com/be91d86ed0ce44f3a4a09eddd20593f7

Porter, J. (2016). "4 Benefits of Learning Programming at a Young Age." *ELearning for Kids*. Retrieved from: https://elearningindustry.com/4-benefits-learning-programming-at-a-young-age-2

Schmidt, E. (2012). "Britain's Economy Will Thrive if Computing Becomes Child's Play." *The Guardian*. Retrieved from: http://www.guardian.co.uk/commentisfree/2012/apr/08/eric-schmidt-improve-computer-education

Singer, N. (2017). Education Disrupted How Silicon Valley Pushed Coding into American Classrooms. New York Times. Retrieved from: https://www.nytimes.com/2017/06/27/technology/education-partovi-computer-science-coding-apple-microsoft.html

Stenger, M. (2017). Coding in Education: Why It's Important and How It's Being Implemented. informed Open Colleges. Retrieved from: https://www.opencolleges.edu.au/informed/features/coding-education-important-implemented/

Stross, R. (2012). "Computer Science for the Rest of Us." *The New York Times*. Retrieved from: http://www.nytimes.com/2012/04/01/business/computer-science-for-non-majors-takes-many-forms.html

Vlatko, N. (2015). *JAX Magazine*. The countries introducing coding into the curriculum. Retrieved from: https://jaxenter.com/the-countries-introducing-coding-into-the-curriculum-120815.html

Walsh, T. (1992-93). "The Implementation and Evaluation of a Sequential, Structured Approach for Teaching LogoWriter to Classroom Teachers" Journal of Educational Technology Systems (Vol. 21 No. 4).

Walsh, T. (1994). "Facilitating Logo's Potential Using Teacher-Mediated Delivery of Instruction: A Literature Review. Journal of Research on Computing in Education (Vol. 26 No. 3).

Walsh, T. (2013-2017). *Exploring Computer Science with Microworlds EX*. Montreal Quebec, Canada: Logo Computer Systems, Inc. (LCSI).

William, J. (2017). "10 Surprising Skills Kids Learn Through Coding." *We Are Teachers.* Retrieved from: https://www.weareteachers.com/skills-learn-coding/

Wills, B. (2016). The United States of Coding. New America Weekly Edition 132. Retrieved from: https://www.newamerica.org/weekly/edition-132/united-states-coding/

Author Biography

Thomas Walsh Jr. was a classroom teacher at the elementary level for over 35 years, primarily with the Ames Community Schools (ACS). Six years were served as an instructor for high ability T/G grade six students in the ACS Extended Learning Program (ELP) facilitating project work in mathematics including coding instruction in Logo programming. For five years served as an adjunct instructor to high ability elementary and middle school students at Iowa State University's Early Outreach Program (EOP), formerly OPPTAG, providing instruction in Logo coding for developing turtle graphics and basic gaming program procedures. Further information about the author's other international teaching opportunities and publications can be found on the web page @ https://sites.google.com/site/tomwalshjrhome/

In 2018 Walsh participated in the CanCode government initiative providing workshop training to teachers in Vancouver, Canada using MicroworldsEX and MicroworldsJr. Workshops and presentations on Logo coding have been conducted at international math and gifted education conferences.

Early publications on Logo programming, based on dissertation research, was conducted and supported teaching of coding to students and teachers using the original Logo version, LogoWriter, and MicroworldsEX for the last 30 years. Hopefully, opportunity to teach Logo instruction using the Lynx program will be provided to enthusiastic coder learners in the future!

Teacher Lesson Plans

Lesson 1: Introduction to Lynx Procedures and Turtle Commands

Objective 1: Given a demonstration of the Lynx platform layout, students will launch a Work Area (Page) with adding a turtle object for typing primitives in the Command Center.

Objective 2: Students will type primitives in the Command Center to draw a geometric shape graphic with the turtle.

Time Period: Three or more 60-minute periods

Programming Guide Sections:
1. *Navigating the Lynx Platform*
2. *Getting Started with Lynx Basic Techniques to Get You Started* (Resource Materials User Guide)
3. *Drawing Turtle Graphics*
4. *Changing Pensize, Graphic Color and the Fill Command*
5. Appendix Resources: *Turtle Primitive Flashcard Cut Outs*, *Turtle Primitives*, *Turtle Degrees* and *Turtle Degree Clock*
6. *Lynx Observation Form (Lessons 1-3)*

Procedures:
1. With a Smartboard or LCD projector, demonstrate how to begin the Lynx program and hatch one turtle in the Work Area (Page). Refer students to *Navigating the Lynx Platform* section of the guide while highlighting the iconic symbols as well as the Procedure Pane, Clipart Pane, Project Tree and Command Center.
2. Optional: Assign students to complete the *Turtle Degree Clock* activity. If practice is needed learning degrees complete the *Turtle Degrees* activity pages.
3. Introduce the turtle primitives using the *Turtle Primitive Flashcards* and post for display.
4. Play "Turtle Commander", directing students to move forward (`fd`) and back (`bk`) shoe size distances (for example, `fd 2` is two foot length steps forward). Direct students to practice turns by commanding `rt 90, lt 45, rt 180, rt 270`, and so on.
5. Refer students to read the sections on *Drawing Turtle* Graphics, and *Changing Pensize, Graphic Color, and the Fill Command* for information on turtle primitives and ideas (for example, *Turtle Hint*!).
6. Provide opportunity for students to practice typing in the Command Center and take notes, if needed, with the *Turtle Primitives* activity sheet.
7. Lynx Program Project - Colors: Assign students to select a turtle activity project based on their interest and ability. Provide the *Turtle Degrees* Clock to students requesting support in learning turtle turns using degrees.

Evaluation: Drawing with the turtle in creating one or more geometric shape graphics using color. Provide student feedback using the *Lynx Observation Form*.

Lesson 2: The Repeat Command and Geometric Shapes

Objective 1: Students will learn and use the `repeat` command to create geometric graphic shapes.

Objective 2: Students will apply the `repeat` procedure to create geometric shapes based on learning the relationship between number of degree turns and the number of repeats.

Time Period: Two or more 60-minute periods

Programming Guide Sections:
1. *Repeat It*!
2. Appendix Resources: *Turtle Primitive Flashcard Cut Outs, Turtle Shapes*, *Turtle Degree Clock, Repeat Predictions*, and *A-Mazing*
3. *Lynx Observation Form (Lessons 1-3)*

Procedures:
1. With a Smart Board or LCD projector, demonstrate the following procedure as follows:
```
fd 100
rt 90
fd 100
rt 90
fd 100
rt 90
fd 100
rt 90
```
   Pose student questions: Do you see a pattern repeating in the list of commands? How many times does the pattern repeat? Then write the following procedure:
```
repeat 4 [fd 100 rt 90]
```
   Pose additional questions: Will the turtle draw the same thing using the list of commands as with the `repeat` instruction? What geometric shape will the turtle draw? Check to see if students make a connection between the list of commands and the repeat instruction. Refer students to guide section for additional information or review the section titled *Repeat It*!
2. Lynx Program Project - Repeat: Assign students to select a turtle activity project based on their interest and ability. Provide a turtle clock to students requesting support in learning turtle turns and degrees.
3. Direct students to work in cooperative pairs to complete the *Turtle Shapes* activity. Ask students to tell the turtle rule or relationship between the number of turns (`rt` or `lt`) and the `repeat` number.
   Rule: Repeat number times the `rt` or `lt` input number = 360 degrees or 360 divided by the `repeat` number = number of `rt` or `lt` degrees.
4. Optional: Assign students to complete the *Repeat Predictions* and/or *A-Mazing* activity.

Evaluation: Completion of one or more geometric shape graphics using the repeat procedure. Provide student project feedback using the *Lynx Observation Form*.

Lesson 3: Introducing Turtle Programs

Objective 1: Students will learn how to write a program using turtle commands (primitives) to create a realistic graphic with geometric features, add a button, and an additional page for a project.

Objective 2: Students will follow the cognitive monitoring planning procedure to plan a graphic, decompose the shapes, write a plan, write program procedures, test the graphic result, draw the initial graphic outcome, and debug the program to work as planned.

Time Period: Three or more 60-minute periods

Programming Guide Sections:

1. *Introducing Turtle Programs*
2. *Getting Started with Lynx Basic Techniques to Get You Started* (Resource Materials User Guide)
3. Appendix Resources: *Turtle Primitive Flashcard Cutouts, Cognitive Monitoring Planning, Turtle Degree Clock,* and *Changing Procedures and Predicting Skills*
4. *Lynx Observation Form (Lessons 1-3)*

Procedures:

1. Review the turtle primitives using the *Turtle Primitive Flashcards* and post for display.
2. With a Smart Board or LCD projector, demonstrate how to write a turtle program on the Procedure Pane and run it in the Command Center for display on a Work Space (Page).
3. Show how to add a button on the Work Space (Page) to run a procedure. Demonstrate adding a button by selecting from the plus "+" symbol, label the button with a name (for example, box), and then type or select the program name (for example, square) in the On click space. Press the button to see how it works.
4. Demonstrate how to add a new page to a project. Point out the left and right arrows (< >) located at the top of the Procedures Pane, for typing the name of the project, can be clicked to switch pages after choosing Page in the "+" menu. Review clicking on the Procedures Pane side triangle arrow to display Procedures and Add a tab option for entering additional program coding procedures.
5. Refer students to the *Introducing Turtle Programs* section of the guide with example figures for additional information about adding buttons. *Getting Started with Lynx Basic Techniques to Get You Started* provides ideas for adding buttons and pages.
6. Lynx Program Project - Procedures: Assign students to select a programming project based on their interest and ability.
7. Direct students to work individually or in cooperative pairs to plan a turtle graphic using cognitive monitoring procedures. Refer to the example glasses project and debugging steps shown in the appendix.
8. Optional: Assign students to complete the *Changing Procedures and Predicting Skills* activity.

Evaluation: Student writes a Lynx program and successfully executes (runs) the turtle program name showing the turtle graphic outcome. Provide student project feedback using the *Lynx Observation Form.*

Lesson 4: Creating Modular and Recursive Programs

Objective 1: Students will learn how to write a modular program using turtle primitives to create a realistic graphic with geometric features for a coding project.

Objective 2: Students will learn how to write a recursive program using turtle primitives to create a realistic graphic with geometric features.

Time Period: Three or more 60-minute periods

Programming Guide Sections:
1. *Creating Modular Programs*
2. *Simple Logo Recursion*
3. *Lynx Rubric Evaluation (Lessons 4-8)*

Procedures:
1. Direct students to read the *Creating Modular Programs* section of the guide and study the example program procedures and figures. Pose questions: How is a basic turtle program different from a modular program?
2. Study the program examples in the guide identifying which programs are subprocedures for modular program superprocedures.
3. Lynx Program Project – Modular Procedures: Assign students to select a modular program project based on their interest and ability.
4. Direct students to read the *Simple Logo Recursion* section of the guide and study the example figures for additional information and procedures. Discuss and identify which programs in the guide show modular procedures and recursion.
5. Provide opportunity for students to write and run recursive programs.
6. Lynx Program Project – Modular Recursive Procedures: Assign students to select a recursion and/or modular program project based on their interest and ability.

Evaluation: Student writes a Lynx program and successfully executes the turtle recursion and/or modular program showing the turtle graphic outcome. Begin to provide feedback to students about their projects using the *Lynx Rubric Evaluation*.

Lesson 5: Assigning Variables in Logo Programming

Objective 1: Students learn how to assign variables in subprocedures and superprocedures for a coding project.

Objective 2: Students will learn how to write a modular program that contains variables, using turtle primitives to create a realistic graphic with geometric features.

Time Period: Six or more 60-minute periods

Programming Guide Sections:
1. *Assigning Variables in Logo Programming*
2. *Recursive Variable Modular Procedures in Logo Programming*
3. *Lynx Rubric Evaluation (Lessons 4-8)*

Procedures:
1. Direct students to read the *Assigning Variables in Logo Programming* section of the guide and study the example program procedures and figures. Pose questions: What is a variable and what is the purpose for use in a program? Where are variables written in the program? How do you test or execute the program to see if the variable works?
2. Study the program examples in the guide identifying the purpose of the variables shown on the procedure lines.
3. Lynx Variable Program Project: Assign students to select a variable program project based on their interest and ability.
4. Direct students to read *Recursive Variable Modular Procedures* in the *Logo Programming* section of the guide and study the example program procedures and figures. Pose questions: What are some procedures than can be used to control recursive variable programs? How can you write a modular variable program? How is a variable coding project run with values (numbers) typed in the Command Center (for example, `shoe 45 78 13`) different from a variable program with assigned values?
5. Project Development Idea: Suggest students write a modular program without variables first to see if the procedures run without errors. Next, have students add one or more variables to the program procedures.
6. Lynx Modular Variable (Recursion) Project: Assign students to select a modular variable program project activity based on their interest and ability.

Evaluation: Student develops a modular program coding project with variable(s) and successfully runs the program procedures. The project variables can have assigned values in the program procedures. The *Lynx Rubric Evaluation* provides feedback to the student for their projects.

Lesson 6: Animating Turtle Shapes with a Slider and Adding Features

Objective 1: Students will write an animated program code, using turtle shapes, to display moving graphics started with a button tool.

Objective 2: Students will write a code to animate shapes moving at varying speeds using a slider or selected program procedures (for example, `forever` with `speed` primitives).

Objective 3: Students will add an additional feature to their project (for example, a background, sound or music and/or a clickable turtle shape).

Time Period: Three or more 60-minute periods

Programming Guide Sections:
1. *Animating Turtle Shapes*
2. *Additional Features for Project Development*
3. *Getting Started with Lynx Basic Techniques to Get You Started* and *List of Lynx Primitives* (Resource Materials User Guides)
4. Appendix Resources: *Multiple Turtles*
5. *Lynx Rubric Evaluation (Lessons 4-8)*

Procedures:
1. Assign students to practice the commands and procedures for the *Multiple Turtles* activity. Guide students to record outcomes on the lines provided.
2. Guide students to read and study the pages in the guide on *Animating Turtle Shapes, Adding a Button and Slider for Animation of Shapes, Animation Procedures with Varying Shape Speed and Added Background,* and *Additional Features for Project Development*.
3. Demonstrate the animated Rundog program and how to select shapes, as shown in the section on *Animating Turtle Shapes*. Pose the following challenge: Predict what is the purpose of the `setsh` and `wait` commands?
4. Discuss the `Lynxrace` program example in the guide on *Adding a Button and Slider for Animation of Shapes*. Pose the following question: What is the purpose of the `forever` primitive?
5. Lynx Animation Program Project: Direct students to work individually or in cooperative pairs to create an animation project with selected shapes created with Lynx tools (i.e., buttons and slider). Encourage students to create a `stopall` button to stop moving turtles on the screen.
6. Provide resource access to the *List of Lynx Primitives* in the Help- User Guides.

Evaluation: Students write an animated shapes program, with varying speeds including additional project features, and successfully executes the project for demonstration. The *Lynx Rubric Evaluation* provides feedback to the student for their projects.

Lesson 7: Going Further: Words and Lists in Logo Procedures

Objective 1: Students will learn commands on how to manipulate words and lists to enter character strings and program procedures for showing outcomes.

Objective 2: Student tests lists and numbers programs to create their own interactive program procedures.

Time Period: Three or more 60-minute periods

Programming Guide Sections:
1. Going Further: Words and Lists in Logo Procedures
2. *Interactive Lists and Numbers Programs*
3. *Getting Started with Lynx Basic Techniques to Get You Started* and *List of Lynx Primitives* (Resource Materials User Guides)
4. Appendix Resources: *A Turtle Calculator Application* (Optional)
5. *Lynx Rubric Evaluation (Lessons 4-8)*

Procedures:
1. Direct students to read *Words and Lists in Logo Procedures* and subtitle sections of the guide studying the example procedures ahead of time.
2. Give students opportunity to test the procedure examples on the computer, providing guided questioning and teacher scaffolding when needed. Alternately, the teacher may prefer to demonstrate the program procedures on the computer to the class for viewing, questions, and discussion. Add a text box from the "+" sign to a Work Area (Page) to display character strings (for example, words and sentences).
3. Lynx Interactive Lists and Numbers Program Project: Assign students to select a program project based on their interest and ability. Provide differentiation for some students capable of integrating the words and lists (e.g., conversation) code into an existing animation project using programming procedures.
4. Optional: Assign students to complete *A Turtle Calculator Application* activity.

Evaluation: Students writes an interactive words and list program that will successfully execute showing conversation dialogue or output numbers in a textbox. The *Lynx Rubric Evaluation* provides continued feedback to the student for their projects.

Lesson 8: Applying Graphics, Animation, and Interactive List Procedures for Developing Games

Objective 1: Students will develop an interactive game using Logo coding procedures and demonstrate the project to the class. Coding ideas can be developed from program procedure project examples (for example, Quick, Theme Based Activity Cards) incorporated into a newly created game.

Objective 2: Students follow the directions provided in the Lynx User Guides to create a game project.

Time Period: Three or more 60-minute periods

Programming Guide Sections:
1. *Applying Graphics, Animation and Interactive List Procedures for Developing Games*
2. Lynx website at https://lynxcoding.club/ access game programs for review
3. Help tool at Lynx website at https://lynxcoding.club/ access User Guides steps to create games
4. *Getting Started with Lynx Basic Techniques to Get You Started* and *List of Lynx Primitives* (Resource Materials User Guides)
5. *Lynx Rubric Evaluation (Lessons 4-8)*

Procedures:
1. Direct students to read *Applying Graphics, Animation and Interactive List Procedures for Developing Games*.
2. Demonstrate access to the Lynx web site resources for students to review examples of interactive games with access to coding procedures.
3. Show students at the Lynx web site how to find and pdf download from the User Guides Quick, Theme Based Activity Cards or Project Plans to create games.
4. Direct students as needed with their game project development to the references *Getting Started with Lynx Basic Techniques to Get You Started* and *List of Lynx Primitives* (Resource Materials User Guides)

Evaluation: Student develop an interactive game using developed project coding ideas or directions provided in the Lynx User Guides. The *Lynx Rubric Evaluation* provides feedback to the student for their projects.

Teacher Activities Answer Guide and Resources

Turtle Degrees Answer Key

Appendix pages 109-110

Part I.

1) `rt 120` or `lt 240`
2) `rt 210` or `lt 150`
3) `rt 300` or `lt 60`
4) `rt 30` or `lt 330`
5) `rt 90` or `lt 270`
6) `rt 270` or `lt 90`
7) `rt 180` or `lt 180`
8) `rt 0` or `lt 360`
9) `rt 60` or `lt 300`
10) `rt 150` or `lt 210`
11) `rt 240` or `lt 120`
12) `rt 330` or `lt 30`

Part II.

1) 3:00
2) 4:00
3) 2:00
4) 10:00
5) 6:00
6) 9:00
7) 6:00
8) 7:00
9) 4:00
10) 1:00
11) 1:00
12) 10:00
13) 8:00
14) 12:00
15) 12:00
16) 5:00
17) 7:00
18) 11:00
19) 5:00
20) 11:00
21) 6:00
22) 11:00

Part III.

1) 3:00
2) 9:00
3) 5:00
4) 9:00
5) 11:00
6) 6:00
7) 12:00
8) 12:00
9) 3:00
10) 12:00
11) 6:00
12) 7:00

Part IV.

1) 90 degrees
2) 150 degrees
3) Four
4) Answers will vary
5) Answers will vary (e.g., picture frame, poster, file cabinent, etc. objects with 90 degree angles)

Lynx Turtle Shapes Answer Guide

DIRECTIONS: For each shape fill in the table using the headings provided. Use the turtle clock, if needed. When finished answer the question at the bottom of the page.

Shape	Number of Sides	Number of Degrees for Each Turn	Repeat Statement
Triangle	3	120	Repeat 3 [fd 100 rt 120]
Quadrilateral	4	90	Repeat 4 [fd 100 rt 90]
Pentagon	5	72	Repeat 5 [fd 100 rt 72]
Hexagon	6	60	Repeat 6 [fd 100 rt 60]
Septagon	7	51.43	Repeat 7 [fd 100 rt 51.5]
Octagon	8	45	Repeat 8 [fd 100 rt 45]
Nonagon	9	40	Repeat 9 [fd 100 rt 40]
Decagon	10	36	Repeat 10 [fd 100 rt 36]
Circle	infinite	1 or 360	Repeat 360 [fd 1 rt 1]
*Other? ____________	vary	vary	Repeat procedure will vary

Question: What is the turtle rule or relationship between the number of turns (rt or lt) and repeat number?

Number of turns (sides) x Number of degrees = 360 degrees

Lynx Observation Form (Lessons 1-3)

Name ____ Date ________________________

Directions: The observation may be used during student project development (i.e., mini-conference) or as a post evaluation. Checked (√) items are skills demonstrated by the student with the Lynx program.

Graphic or Project Title:

____

Skills: Creating Graphic Designs Demonstrated In:	√
1. Utilizing and controlling drawing and turning commands	
2. Adding a button to a page to link pages and run procedures	
3. Creating geometric shapes using a repeat procedure	
4. Writing and successfully running a program procedure	
5. Other Skill (if any):	

Brief Comments and Suggestions: ____

____

--- cut --

Lynx Observation Form (Lessons 1-3)

Name ____ Date ________________________

Directions: The observation may be used during student project development (i.e., mini-conference) or as a post evaluation. Checked (√) items are skills demonstrated by the student with the Lynx program.

Graphic or Project Title:

____

Skills: Creating Graphic Designs Demonstrated In:	√
1. Utilizing and controlling drawing and turning commands	
2. Adding a button to a page to link pages and run procedures	
3. Creating geometric shapes using a repeat procedure	
4. Writing and successfully running a program procedure	
5. Other Skill (if any):	

Brief Comments and Suggestions: ____

____

Lynx Rubric Evaluation (Lessons 4-8)

Name ____ **Date** ______________________

Directions: The rubric may be used during student project development (i.e., mini-conference) or as a post evaluation. Classroom peers may use the rubric to evaluate student project. Circle ahead of time, before making classroom copies, the programming focus areas (e.g., graphic modular programming using buttons or words & lists programming with graphic animation uisng a textbox).

Graphic or Project Title: ____

Rating:	Exceeds Expectations	Meets Expectations	Needs Redesign
Graphic			
Creativity	Originality and balance shown	Developing design features	Graphic needs development
Detail/Sophistication	Displays detail in design	Some details evident	Simplistic graphic display
Programming			
Graphic geometric complexity, programming skill applications, and problem-solving elements	Graphic contains complex geometric construction requiring problem-solving programming skills	Graphic shows geometry skills using basic programming skills and problem-solving	Graphic needs further geometry development and use of programming skills
Application of Subprocedure Commmands	Highly developed subprocedures and turtle commands	Subprocedures and commands utilized	Few or no subprocedures are evident
Utilizes Programming Method (modular, recursion, variables, and/or animation)	Sophisticated application of programming procedures	Developing programming procedures	Lacks clearly defined programming procedures
Program Effectively Runs and Operates	Efficiently runs	Most procedures run	Procedure errors and debugging needed
Other Program Features			
Tools: Buttons, Music, Hyperlink, Shapes and/or Slider	Creatively applies	Utilized in project	Not effectively utilized
Textbox with Words and Lists Use	Effectively applies	Utilized in project	Not effectively utilized
Other (identify): __________________	Supports and enhances project	Utilized in project	Not effectively utilized

Overall Project Rating (Circle): Exceeds Expectations Meets Expectations Needs Redesign

Comments: ____ (use back, if needed)

Turtle Primitive Flashcard Cut Outs*

*Option: Post cards on a bulletin board for student reference

Drawing and Movement Commands

Pd
Pu
Pe

Home
Setpos [x y]

Drawing Commands

Fd n

Bk n
Setc n
Fill

Setbg n
Setpensize n
Repeat n [fd n rt n] **Repeat n [bk n lt n]**

Turning Commands

Rt n
Lt n
Seth n

Turtle Viewing and Changing Shape

Ht / St
Setsh n

Cleaning and Erasing Commands

Cg

Ct

Turtle Reporters

Show Heading

Show Pensize

Show Pos
Random n

Print 'Word'
Print [A list of words]
Print (78 + 321 + 55) / 4

Print :Word *Inputs word in a program procedure
if :size > *n* [stop] *if [this is true] [then do this action]

Section 3: Introducing Lynx in Eight Lessons

Navigating the Lynx Platform

Layout Windows Design

To get started using Lynx go to the website at https://lynxcoding.club/. Read the cover page followed by viewing the *video The Missing LINK* for an overview of the program. Select the button CREATE A LYNX PROJECT to begin free limited access to the program (note **Lynx Use Policy** on the next page). You will see a screen like the following shown in Figure 1.

A view of the Lynx working environment shows three main window spaces. The bottom window called the **Command Center** is used for experimenting with turtle commands (for example, Logo primitives like `fd 100`). The program-coding procedures are typed in the **Procedure Pane** side window (for example, the program named `laptop`). The main window called the **Work Area (Page)** is where you can hatch turtles for display of graphics and animated projects created by the program procedures. Refer to figure 1 to view the layout of the Lynx working environment and access to the tool icons.

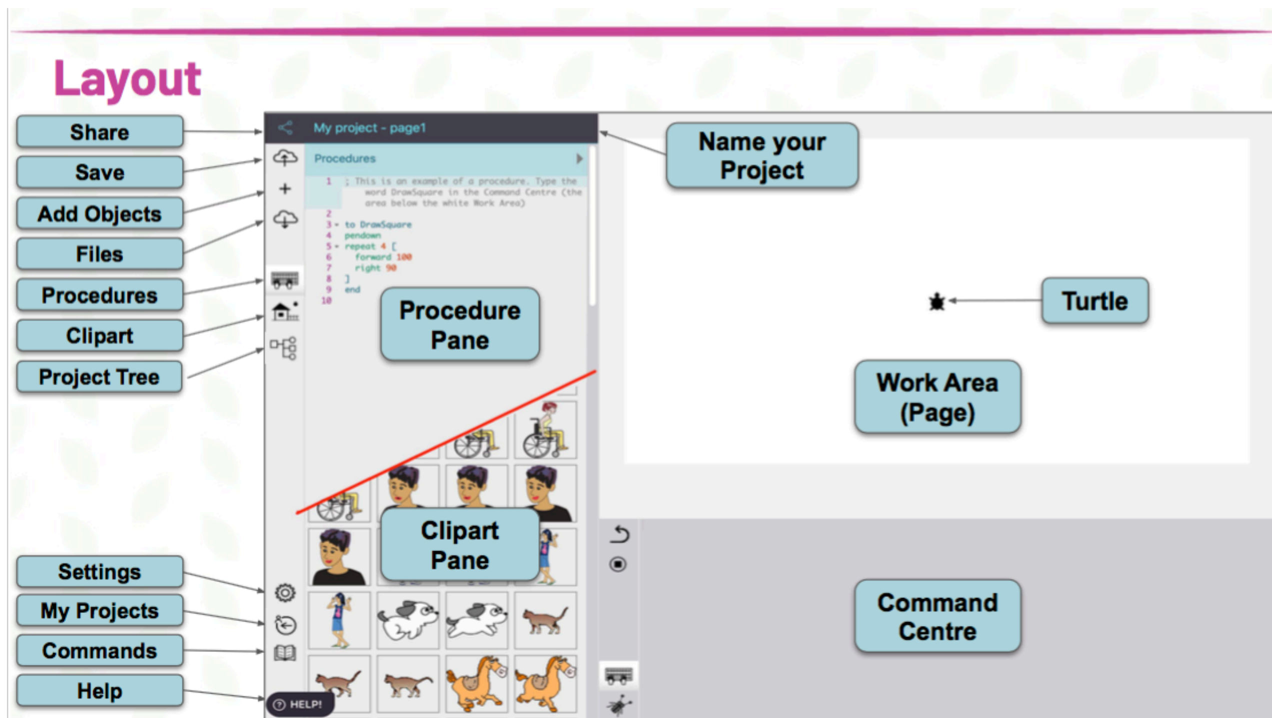


Figure 1. Lynx layout features available from the Ecosystem activity and also from other cards on the Lynx web site.

Descriptions of the layout features is provided in the following bullet points. Notice along the Procedure Pane are symbols to support the development of program projects. A short explanation of the icon features include:

- Up arrow cloud: To upload and save a project to the cloud (Login required)
- Plus sign "+": For adding a turtle, text box, button, slider, hyperlink, sound, page and simple clipart to a project.

- Down arrow cloud: To retrieve and import samples of existing and new projects.
- Keyboard: Return to typing program codes in the procedures window.
- House icon: Residence for adding shapes and sample clipart from the “+” sign.
- Project tree icon: Place to see and manage pages with objects for a project.
- Gear: Window display for selecting font and mode or level of learning.
- Circle left arrow: Permits exiting a page.
- Book icon: Displays Lynx Learner Mode Help window of turtle primitive commands and some technical information.
- HELP! Button: Quick answers for common problems by entering search for help questions.

Also shown are some icons displayed next to the Command Center window used for testing turtle primitives and developing program procedures. These symbols include:

- Curve left arrow: Go back to the previous command or **undo** a turtle procedure move.
- Square inside circle: A stop button to end a continued turtle movement activity (for example, a recursive or animation program).
- Keyboard: Return to typing program procedures in the command center window.
- Insect icon: For use in developing coding programs using variables.

Left and right arrows (< >) located at the top of the Procedures Pane, for typing the name of the project, can be clicked to switch pages after chosing Page in the “+” menu. Click on the Procedures Pane side triangle arrow to display Procedures and Add a tab option for entering additional program coding procedures. Refer to page 85 for additional information about adding pages to a project. As pages are added the **project tree** is useful to see and manage objects for a project. It is useful for tablet users when right-clicking is not available and to manage objects located in different pages in one location. It is also helpful to find a turtle or change a text box from invisible to visible. Refer to *Getting Started with Lynx* page 33 for further instructions on managing the project tree.

The **Help** tool at the top of the Lynx website has a pull down window to select **User Guides**. The downloadable pdf documents provide a layout graphic page displaying the symbols and tool features on a Lynx page. For example, refer to the document on *Ecosystem* on page 5 to view the Layout features.

User Guide Support: Getting Started and List of Primitives

Additional information about the Lynx program to support navigation and program project development is provided in the guide *Getting Started with Lynx Basic Techniques to Get You Started*. Download the guide at the Help pull down window. The Help pull down window is displayed at the top to the Lynx website page. Scroll down to Resource Materials and select the Getting Started link for a pdf copy of the text. References to the guide will be made in the e-book to provide support in use of the Lynx program.

Students appreciate reference access to Logo commands. Access the *List of Lynx Primitives* user guide from the Help tool at the top of the Lynx website. Pull down window to select User Guides. Then scroll down the page and select the link List of Lynx Primitives from the Resource Materials. The downloadable pdf documents provides an alphabetical list of Logo primitive commands and reports the turtle movements with examples. When typing the name of a primitive in the Command Center or in the Procedures Pane you can hover your mouse pointer

over the name of the command for a brief description and example. For example, typing `repeat` displays the following window shown in Figure 2.

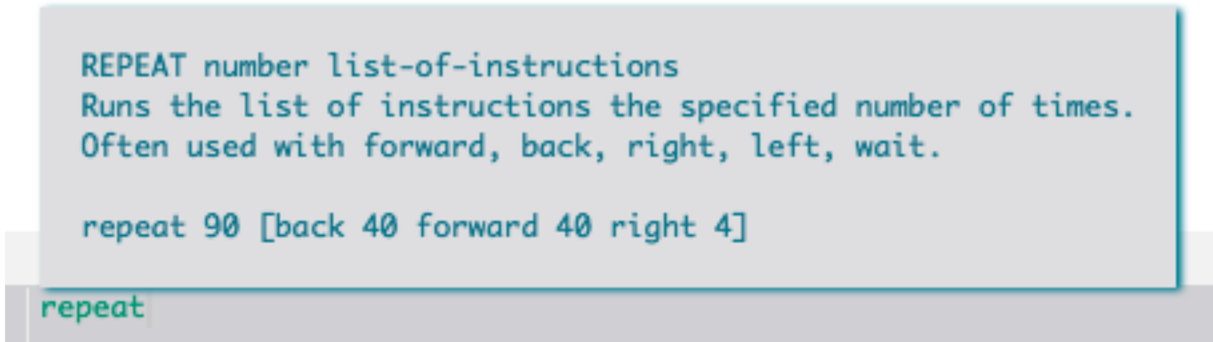


Figure 2. Window display example for `repeat` primitive when hovering over command.

Lynx User Policy

Lynx can be used for free without creating an account limiting usage of all the features of the program including saving projects. To register with Lynx to gain full program access select the Help pull down window. Remember the Help pull down window is displayed at the top to the website page. A screen shot of a project can be taken by pressing Control-Command keys to open a window and selecting Take a Screenshot. Coding program procedures can be copied and pasted to another file (example: Word document) for viewing or reference. User Guides for pdf download information is provided by selecting the following links:

- How to Create a Free Trial Account
- How to Convert a Free Trial Account to Permanent Individual
- How to Create and Manage a School Account

Other links are also shown on the page for viewing.

Saving Projects and Sharing with Friends

Refer to *Getting Started with Lynx* pages 30-32 for instructions on saving and retrieving your project and sharing your project with friends. The authors of the publication remind users **there is no autosave with the Lynx** program so it is important to save your project as your are working on it before leaving the project editor. If a student sees a red dot beside the *Up-to-the-Cloud* icon, it is time to save! It is not uncommon for students to lose projects after investing much time when working on program procedures.

Drawing Turtle Graphics

To begin Lynx to create graphics, with a turtle in the middle of the workspace (in the 'home' position), hatch a turtle by selecting from the "+" symbol on the menu side bar. Type turtle instructions in the Command Center window (at the lower part of the screen) and the turtle draws on the page.

Words built into Logo's vocabulary are called primitives. Turtle primitives are not case sensitive and may be typed in the command center in capitals or lower case letters. Some of the words, or primitives, that can be typed in the Command Center to move the turtle are `forward, back, right, left, pd,` and `pu`. `Forward` or its short form, `fd`, moves the turtle ahead the number of units (turtle steps) typed. `Fd 50` moves the turtle `forward 50` turtle steps and, if you've put the turtle's pen down (by typing `pd`), it will draw a line 50 units long. `Back` (shortform is `bk`) makes the turtle move backwards. `Bk` 100 moves the turtle backward and, if the pen is down, draws a line 100 units long.

The `right (rt)` and `left (lt)` commands turn the turtle the number of degrees typed after the command. Examples of these commands are:

`rt 90`	The turtle turns right 90 degrees, making a corner.
`lt 90`	The turtle turns left 90 degrees, making a corner in the other direction.
`rt 180`	The turtle turns around right 180 degrees and is ready to move the opposite direction.
`lt 45`	The turtle turns left 45 degrees and is ready to draw at an angle.

`Fd, bk, rt,` and `lt` all require input – a number that provides additional information. Always put a space after the Logo commands `fd, bk, rt,` and `lt` and then type a number. Press the **Return/Enter** key after typing a command to instruct Lynx to run the command.

Other useful commands are `cg` (clear graphics) and `cc` (clear commands), which clears the Command Center. The `cg` command clears all the graphics and puts the turtle in the center of the screen, pointing up (in the 'home' position). Type `cc` to clear the Command Center. This erases all the commands typed into the Command Center.

There are other commands that can be typed into the Command Center and used to draw pictures with the turtle. To lift the turtle's pen and have the turtle move on the screen and not draw a line, type `pu` (for pen up). For example, to move the turtle `forward 20` turtle steps and not draw a line, type `pu fd 20`. To put the turtle's pen down again for drawing, type `pd` (for pen down). For example, to draw a line 20 units long, type `pd fd 20`.

Additional Primitive Drawing Commands

Using many `rt` and `lt` commands may cause confusion – it will be hard to tell in which direction the turtle is going or heading at the end of all the instructions. The instruction `show heading` gets Lynx to report the turtle's heading in degrees from 0 to 359. When the turtle is pointing straight up, `show heading` reports 0. In the Command Center, type:

```
cg rt 45 show heading
```

Press **Return/Enter** and the number 45 is printed. To turn the turtle back to facing the top of the screen, type `home` in the Command Center and press **Return/Enter**.

The command `seth` (set heading) with an input gets the turtle to turn to a specific direction based on compass positions. `Seth 180` always turns the turtle so that it points to the bottom of the screen. Seth 90 always turns the turtle so that it points to the right of the screen. `Seth 270` always turns the turtle so that it points to the left of the screen.

Using the `pu` command with `setpos` (set position) is an good way to move the turtle to different screen locations when starting a project. The input to setpos is always two numbers in square brackets. The numbers in the brackets are *x* and *y* screen coordinates. For example, the center of the screen (the 'home' position) is [0 0] so `setpos [0 0]` positions the turtle in the center of the screen. The reporter `pos` reports the turtle's position (its *x* and *y* coordinates). To have the position printed in the Command Center, type: `show pos`. Remember setpos needs two numbers in the square brackets (not parentheses) as input for it to work.

Another interesting command is `pe` (for pen erase), which can be used to erase a line. For example, draw a line using `pd fd 100`. Next, type in the Command Center: `pe bk 50`. This erases half the line. When using any of the pen commands (`pe, pu, pd`) you need to use a command that moves the turtle, such as `fd` or `bk`, and a number to see their effects. For example, try:

```
pd fd 100
pe bk 100
```

Other commands you may want to try are `ht` (hide turtle) and st (show turtle). To draw lines with an invisible turtle, type `ht` before drawing. To see the turtle again type `st`. The `ht` command can also be used with the `pu` command to move the invisible turtle around the screen without seeing it. The `ht` command is sometimes used at the end of a project when you do not want the turtle to be part of the scene.

Learn turtle command functions when typing Logo primitives in the command center. To quickly get a short definition and example of each primitive in a message box, let your cursor hover over the primitive in the command center or Procedures Pane.

For a more detailed explanation, select the **book** symbol on the menu bar to learn more about turtle primitives. Select **HELP!** to search for information about procedures and Logo primitives. Access the *List of Lynx Primitive* user guide from the Help tool on the Lynx website at https://lynxcoding.club/. Also refer to *Getting Started with Lynx* pages 30-32 for instructions on saving and retrieving your project and sharing your project with friends.

Turtle Hint!
A quick way to move the turtle around the screen without drawing lines is to click (with the mouse or pad) on the turtle and, with the mouse button held down, drag it any position in the workspace.

Turtle Hint!
After adding a turtle from the "+" object on to a Work Area (Page) remember to type `pd` (for pen down) in the command center first, in order for the turtle to draw. The turtle will continue to draw using the command unless you type `pe` (for pen erase) or `pu` (for pen up). After using the `pe` or `pu` commands remember to type `pd` again.

Turtle Hint!
A technique for saving time when typing in the Command Center is to use the up and down arrow keys. These keys scroll the commands in the Command Center up and down. By placing the cursor over an instruction you used earlier and pressing **Return/Enter**, the turtle will repeat that instruction. This is helpful because it means you don't have to retype commands. For example, if `st pd bk 100` has been typed into the Command Center and you want to repeat the command, press the up arrow key to move the cursor back to the original line, then press the **Return/Enter** key. Now, you've drawn the line an additional 100 units backward. You can also use the `cc` (clear commands) to erase all primitives typed in the Command Center.

Lynx Program Project – Turtle Commands

Turtle Activity 1
Predict the graphic each set of commands will make and then try the following instructions. Was your prediction correct?

1. `Cg pd fd 50 rt 90 fd 50 rt 90 fd 50 rt 90 fd 50`
2. `Cg pd lt 90 bk 90 rt 45 fd 60 lt 90 fd 60`
3. `Cg setpos [-200 150] pd rt 45 fd 1000 lt 90 fd 1000`
4. `Cg pd rt 90 fd 20 pu fd 20 pd fd 20 pu fd 20 pd fd 20 pu fd 20 pd fd 20 pu fd 20 pd`
5. `Cg pd fd 100 pe bk 20 rt 90 pd fd 30 bk 60`

Turtle Activity 2
Make geometric shapes (polygons) with these primitives: `fd, bk, rt,` and `lt`. Create two or more of the following shapes: a square, rectangle, triangle, hexagon, or octagon.

Turtle Activity 3
Make one of the following graphics with the turtle primitive commands: flag, house, rocket, or star. In addition, create a street scene using the `pu` and `pd` primitive commands. *Hints & Suggestions:* The `pu` and `pd` commands are helpful in creating the dashed center line on a road. Create buildings with the turtle primitive commands.

Changing Pensize, Graphic Color, and the Fill Command

The width of the turtle's pensize can be changed using the `setpensize` command followed by a number from 1 (standard) to 30 (thickest). The color of the turtle and it's pen can be changed by typing the command `setc` (set color) with a color number as input. For example `setc 105` will change the turtle and it's pen color to blue. `setc "blue` will do the same. `Setc 9` returns the turtle color to black.

The color of the workspace background can be changed by typing `setbg` (set background) with a color number as input. For example, `setbg 65` will change the background to green. `Setc 'green'` or `setc "green` will do the same. `Setbg 0` will return the workspace to its original clear appearance. A report of the last background color used is made by typing Show Bg (background).

To see the colors available on Lynx, select the book icon on the menu bar to open the window and then select Other Stuff. On the side menu select the link under Lynx Colors (Lynx color names and numbers). A window will open showing a chart of Lynx Color Names and Numbers with information and procedure code examples.

Another command that will make the graphic pictures more interesting is the `fill` command. This command is used to color in or fill in a shape. Try this program to fill in a square:

```
cg
pd
repeat 4 [fd 100 rt 90]
pu
rt 45 fd 30
pd setc 67
fill
```

Sometimes it gets confusing knowing the pen color of the turtle, especially when drawing over filled graphics and backgrounds. `Show color` reports a number telling the color number of the turtle or filled object. Refer to the example of a student gear project using pensize, color, and fill command (Figure 3).

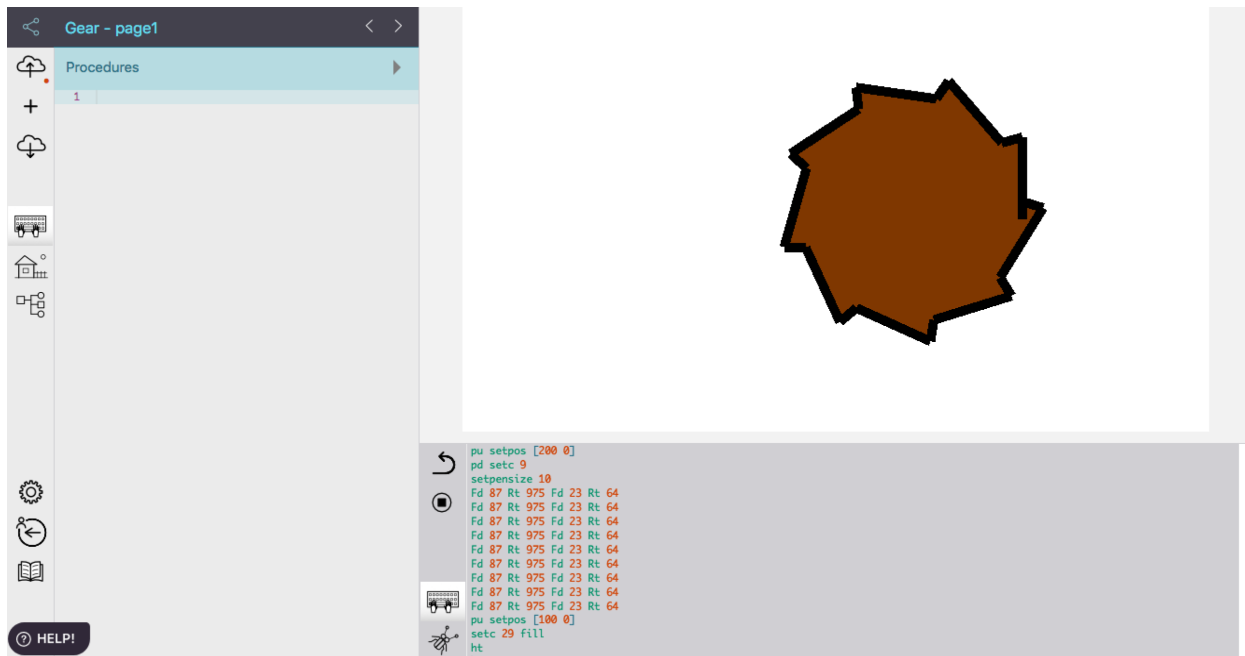


Figure 3. A student gear project using setpensize, setpos, and fill commands.

Turtle Hint!
When you use `fill` to color-in a graphic on the screen, the turtle is the same color as the background and is difficult to see. Try setting the turtle to a different color, for example `setc 9`, in order to see the now black turtle inside the colored graphic.

Lynx Program Project - Colors

Turtle Activity 1
Add color using the `fill` and `setc` commands to a selected previous activity.

Turtle Activity 2
Copy the gear turtle commands below, which are displayed in Figure 2. Create you own gear project by making changes to the turtle procedures.

```
pu setpos [200 0]
pd setc 9
setpensize 10
Fd 87 Rt 975 Fd 23 Rt 64
Fd 87 Rt 975 Fd 23 Rt 64
Fd 87 Rt 975 Fd 23 Rt 64
Fd 87 Rt 975 Fd 23 Rt 64
Fd 87 Rt 975 Fd 23 Rt 64
Fd 87 Rt 975 Fd 23 Rt 64
Fd 87 Rt 975 Fd 23 Rt 64
Fd 87 Rt 975 Fd 23 Rt 64
Fd 87 Rt 975 Fd 23 Rt 64
pu setpos [100 0]
pd setc 29 fill
ht
```

Turtle Activity 3
Create polygon shapes with the turtle using different colored lines for drawing and filling the shapes.

Repeat It!

What Does Repeat Do?
Repeat is a powerful Logo turtle primitive command. The repeat command is used in combination with other Logo primitives. Repeat allows the user to combine groups of commands into a one line procedure. Compare the two columns of commands below. What groups of commands are repeating in the first column? How many times do these commands repeat? What will the repeat procedure make?

Primitives	**Repeat Instruction**
`fd 50  rt 90`	`repeat 4[fd 50 rt 90]`
`fd 50  rt 90`	
`fd 50  rt 90`	

```
fd 50  rt 90
```

Do you see the relationship between the number of times the pairs of commands `fd 50 rt 90` are repeating in the list of primitives and the number four placed after the repeat statement. You can save time using the repeat command to create shapes. In the example above, you can see both a long and a short way to draw a square. Would `repeat 4 [bk 50 lt 90]` produce a similar sized square?

Use the repeat command to create interesting shapes and effects. Begin by typing `repeat` *number_of_times*, the open square bracket, instructions, and the close square bracket. For example, `pd repeat 2 [fd 250 rt 90 fd 150 rt 90]` will make a rectangle 250 by 150 units long. Thus, `fd 250` and `fd 150` draw the lines for the length and width of the rectangle, and rt 90 instructs the turtle to make a 90-degree right turn. Note the use of the repeat command in the picture of the swim goggles (Figure 4).

By changing the inputs, amazing pictures can result. The repeat command can be used to create many polygons and enhanced polyspiral graphics and star shapes. Try: `repeat 5 [fd 75 rt 144]`.

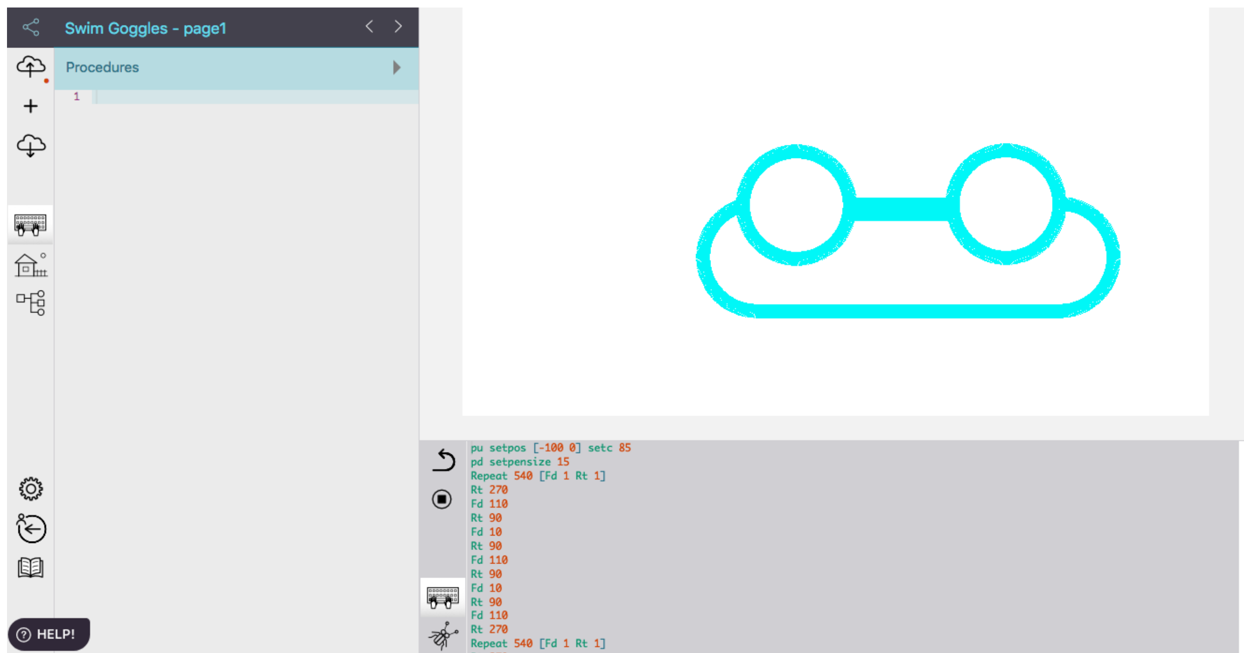


Figure 4. Student swim goggle created with turtle primitives and repeat instructions. (Refer to appendix for program procedures.)

Lynx Program Project - Repeat

Turtle Activity 1

Use the `repeat` command to create different kinds of polygons including shapes with equal and noncongruent sides.

Turtle Activity 2
Use `repeat` for making turtle graphics using a combination of shapes. For example create a house, arrow, rocket, or other object.

Introducing Turtle Programs

Coding in Logo is a process of teaching the turtle new words or commands. You have been using 'built-in' commands (for example, `fd`, `rt`, and `repeat`) that are already part of the Lynx Logo vocabulary. These 'built-in' words are called primitives. An important feature of the Logo language is the ability to teach the turtle *new* commands and words. Once you define a new word, it becomes part of Logo's working vocabulary and can be used in your project just like a primitive.

You teach Logo new words by defining them in terms of words Logo already knows. Word definitions are called procedures. For example, to 'teach' Logo the new command SQUARE, you would type the following procedure in the Procedures Tab in the right side window. Your procedure would look like this:

```
to SQUARE
repeat 4 [fd 40 rt 90]
end
```

First, make sure you have a turtle on the page and that its pen is down. Type the word SQUARE in the Command Center to test the program. When you type SQUARE in the Command Center, the turtle should draw a square (as long as you have correctly typed primitive commands and used spaces correctly). When you name and save the project (for example, **MySquare**), the procedures are also saved. SQUARE will be part of Logo's vocabulary when you open the **MySquare** project, just like all the 'built-in' Logo primitives.

There are three steps to writing a procedure:

`to square`	This naming Line must always begin with 'to'.
`repeat 4[fd 40 rt 90]`	Instruction line(s).
`end`	End line.

Always press **Return/Enter** after 'end'. Examples of students programs are shown with the smileyface and arrow procedures (Figures 5 and 6).

If you want all turtles and buttons to use a Logo procedures write it in the Procedures Pane. Always begin the procedure with the word to followed by a procedure name. The word to with a name lets Logo know that this is the beginning of a procedure. The name must be a single word, with no spaces in it, and cannot be the same name as any of the Logo primitives. The procedure must always finish with the word 'end' and you must always press **Return/Enter** after the word. This tells Logo that you have finished writing the procedure. You will eventually write lengthy programs in the Procedure Pane to create a turtle graphic like the smiley face project (Figure 5).

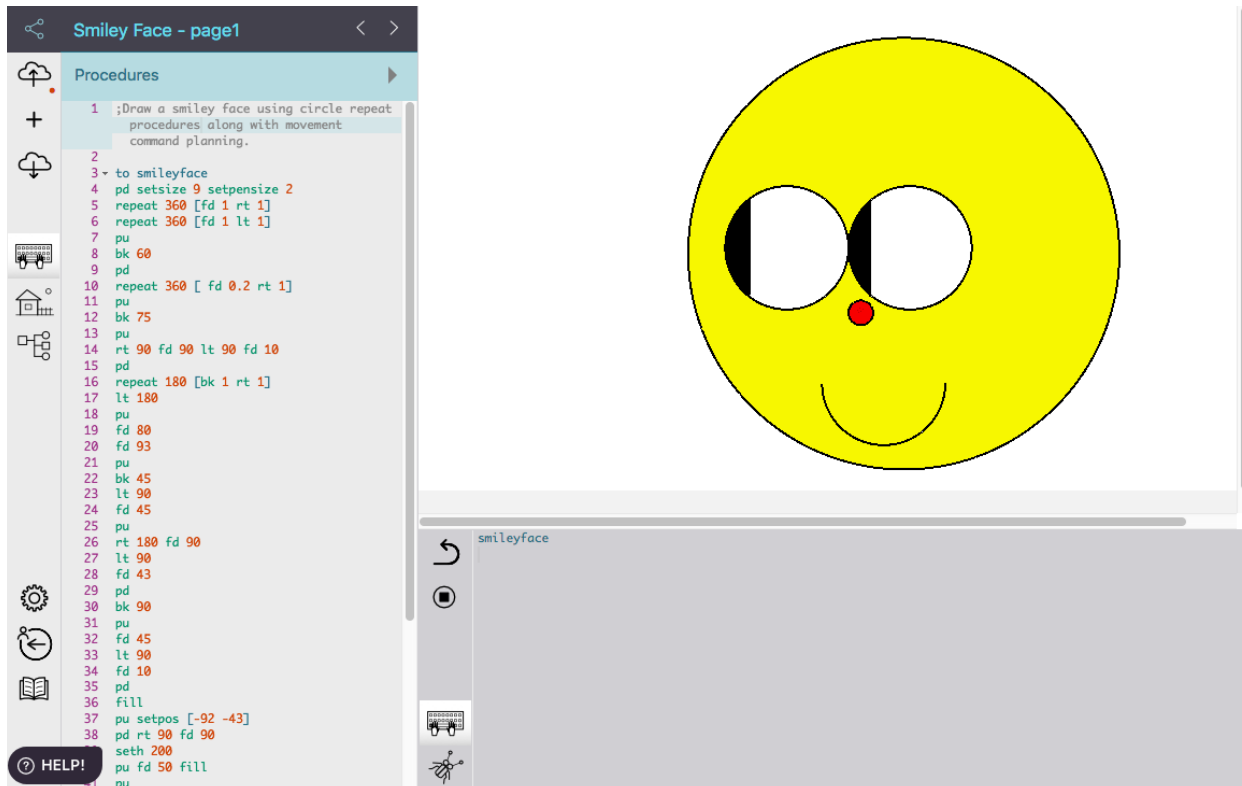


Figure 5. Student smileyface program created with turtle primitives and repeat instructions. (Refer to appendix for program procedures.)

> **Turtle Hint!**
> Type the `pd` (for pen down) command as the first instruction after the naming line in your program procedure. This will allow the graphic to be drawn without having to type `pd` in the command center each time.

> **Turtle Hint!**
> **Learn turtle command functions** by hovering the cursor over the primitive in the procedures panel to see a short descripton of the primitive and an example on how it can be used. Remember you can also select the book symbol or use the HELP! question button on the tool bar to find out more about using turtle commands and procedures.

> **Turtle Hint!**
> **Recommended:** Help yourself, and others, by adding comments in the Procedure Pane before and between programs. You can use words to describe what the programming procedure does. This is good programming practice for self-evaluation and sharing with others. Suggested to add comments starting with a semi-column (;). Note comments written for Figure 5.

Getting Inside the Turtle's Backpack to Run Programs on a Click

If you only want the turtle to use a program procedure write it in *that* turtle's backpack. To get inside the turtle's backpack right click on a turtle to view the Name window showing t1 standing for turtle one. Select the program name (for example, square) in the On click field. Click

on the Apply button to close the window. Click on the turtle several times. What did you make? Refer to page 84 for additional information about a clickable turtle.

Adding a Button to Your Lynx Project

Add a button to your project that will run the program procedures. Begin by chosing Button from the plus "+" menu. Right-click on the button to open the dialogue box and name the button (for example, the name of your program procedure). For the On click drop down menu select the name of the procedure you have created (for example `arrow`) and then click on the Apply button. If you decide to make another button and the name of the program procedure is not displayed in the button window then select New . . . on the On Click drop down menu. The button2_click program will then appear at the bottom of the Procedures pane. After the button2_click naming line write the turtle command to run the desired code outcome (for example `cg`). The Clear Page button procedure is shown as follows:

```
to button2_click
cg
end
```

Examine Figure 6 to view the Arrow and Clear Page buttons added to the page. The arrow button was clicked several times to create more than one arrow graphic design.

Refer to *Getting Started with Lynx* page 15 for further instructions on adding and using buttons. Remember to access the publication from the Help tool at the top of the Lynx website at https://lynxcoding.club/ and pull down window to select User Guides. Then scroll down the page and select the link *Getting Started with Lynx*. You can also access the *List of Lynx Primitives* user guide from the Resource Materials to view or pdf download.

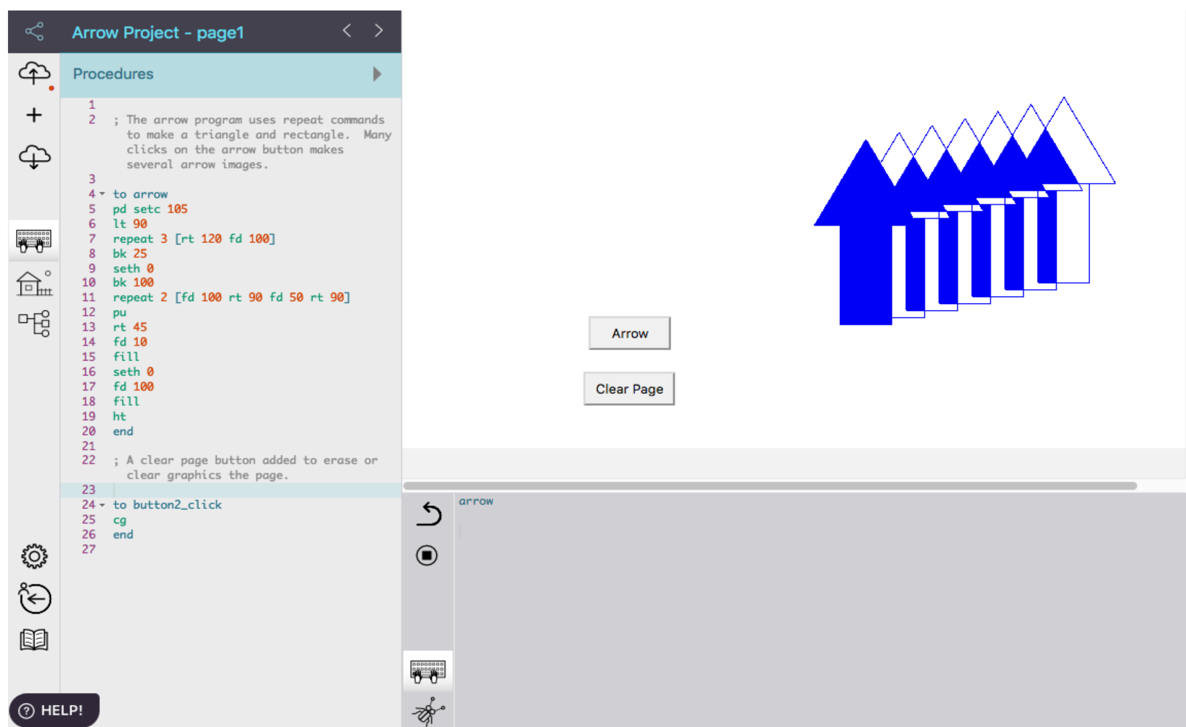


Figure 6. Student arrow program using turtle commands with repeat instructions and added buttons. . (Refer to appendix for program procedures.)

Turtle Hint!
Which programming technique will you use to create turtle projects?
1. **Top-Down –** If you would prefer to go directly to the Procedures Pane in the left side window and type the turtle procedures and then test the new procedure in the Command Center, then you are a top-down programmer.
2. **Bottom-Up –** If you would rather type turtle primitives, one-by-one, in the Command Center, and, once you get a result that you like, copy and paste them into a procedure that you created in the Procedures Pane. Return to the Command Center and test your procedure. If you prefer this method then you are a bottom-up programmer.
3. **Combination –** Some computer programmers use both top-down and bottom-up techniques.

Turtle Hint!
Develop planning skills to mentally think and record comments on the graphic you are attempting to create using the turtle commands for programming procedures. Use the following questions to guide your thinking:
1. What is the final object or project idea you want the turtle to draw?
2. What shapes and repeat procedures will be needed to create this graphic?
3. Will turtle primitives be needed to add color and fill-in shapes?
4. Which turtle commands will be needed to move and position the turtle before typing additional procedures?
5. How can the turtle primitives or procedures be grouped together to build each piece of the drawing to most effectively create your final graphic?

Turtle Hint!
How do I fix and debug my Logo procedure so it will run effectively?
After testing your Logo procedure you may have received one or more error messages. Use the error messages as a guide to edit and change your program in order to create the desired graphic. To make these changes simply copy and paste your program procedures instruction lines (not the 'to' line or 'end' line) from the Procedures Pane to the Command Center to test each line *one by one*. Press the **Return/Enter** key after each line of commands to look for the program error. Once you find the error, edit (or repaste) the instructions into the Procedures Pane. Test the procedure again to see if it works, otherwise debug again.

Lynx Program Project - Procedures

Turtle Activity 1
Create or use Logo commands to write your own turtle procedure showing an object of interest to you. Type the name of the procedure in the Command Center to see if it works as planned. After testing the program successfully add a button to run the program.

Turtle Activity 2
Select from the Appendix a coding project Figure 3 (`SwimGoggles`), 4 (`smileyface`) and/or 5 (`arrow`) to copy and run in the Command Center. Debug and add program procedures to enhance the project.

Turtle Activity 3
Write more than one Logo program using `repeat` instructions to create a procedure that draws an object of interest. Type the name of the procedure in the Command Center to test your program. If the program runs effectively you may be finished, if not use the error message to help debug the program. Add two or more buttons to the program coding procedures.

Creating Modular Programs

Modular programming involves breaking a procedure into parts. The following program shows how to teach Logo the new commands SQUARE and TRIANGLE, typed in the Procedures Tab at the side of the page. Make sure the Pen is Down.

```
to SQUARE
repeat 4 [fd 100 rt 90]
end
to TRIANGLE
repeat 3 [fd 100 rt 120]
end
to HOUSE
SQUARE
fd 100 rt 30
TRIANGLE
end
```

When you teach and tell Logo to SQUARE or TRIANGLE, the turtle draws these shapes. SQUARE and TRIANGLE are **subprocedures** when they are used in another procedure, in this case, HOUSE. A procedure that uses subprocedures, like the HOUSE procedure, is called a **superprocedure**. In this example, the subprocedures SQUARE and TRIANGLE are used to make a HOUSE. Type HOUSE in the Command Center. and the turtle draws a house. Writing programs with subprocedures makes it possible to create more powerful and complex programs, programs that can even instruct the turtle to draw an entire scene with a one-word command.

Study the following modular program that uses the `square` and `triangle` subprocedures:

```
to GuessPic
square
triangle
end
```

Is the `GuessPic` procedure a superprocedure? What graphic will this program make? Did you guess an envelope?

Here is another example of modular programs:

```
to pent
repeat 5 [fd 20 rt 72]
end
to hex
repeat 6 [fd 20 rt 60]
end
```

```
to soccer
pent
lt 120
hex
repeat 4 [lt 132 fd 20 rt 60 hex]
end
```

Which programs are subprocedures and what is the name of the superprocedure in this modular program?

Remember to write comments about your procedures to explain and keep track of your coding programs as they become more advanced. You should start comments on the first line or between changing program lines with a semi-colon (;). For example, refer to Figure 7.

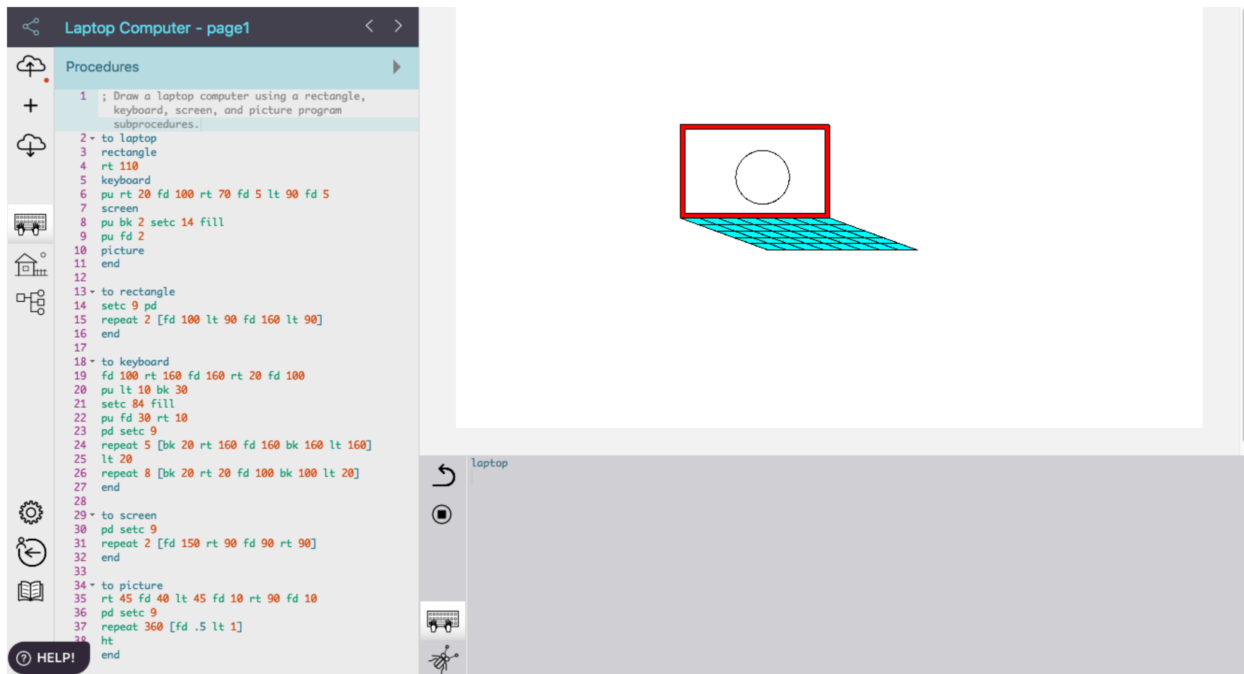


Figure 7. Lynx student project showing a computer laptop program with comments explaining coding procedures. . (Refer to appendix for program procedures.)

A variety of coding strategies can be used to write a modular program. As project graphics develop in greater detail and sophistication students will find it helpful to write smaller sized program subprocedures to use in a super modular program. The smaller coded programs make it easier to find program bugs or coding errors when testing super program procedures to run the entire graphic project. Examples of student modular progrom procedures are provided (Figures 8-11).

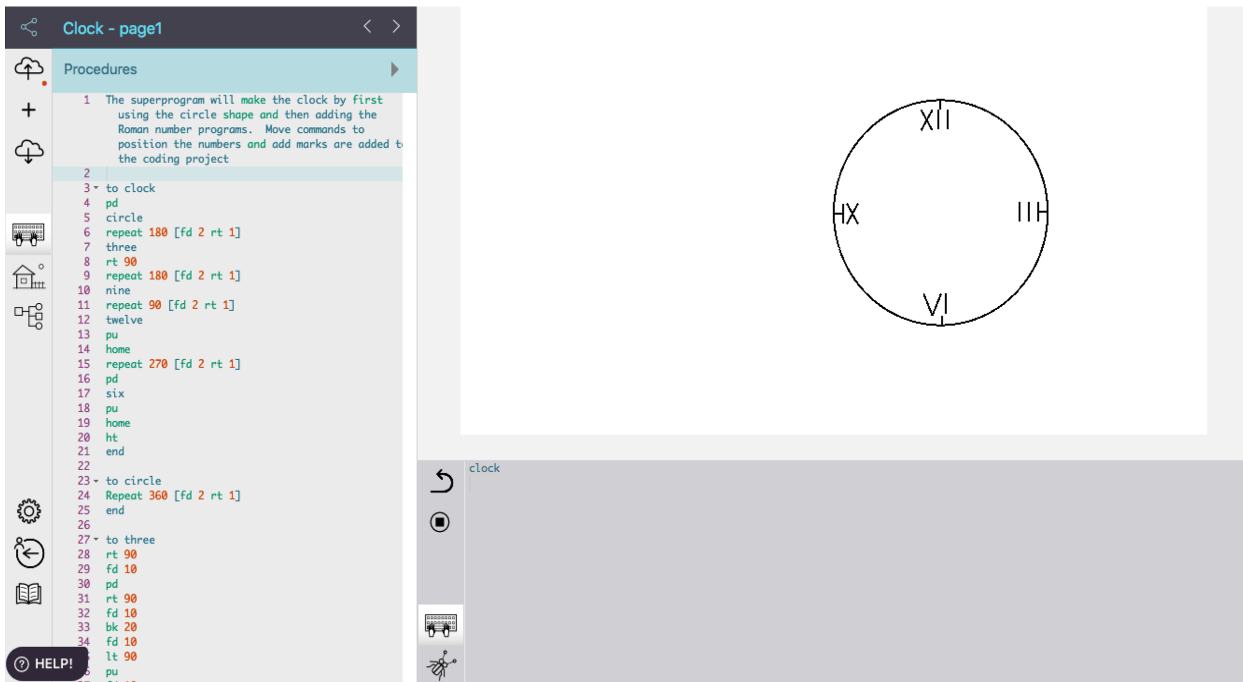


Figure 8. A student modular clock program with circle and number subprocedures. (Program code view in appendix.)

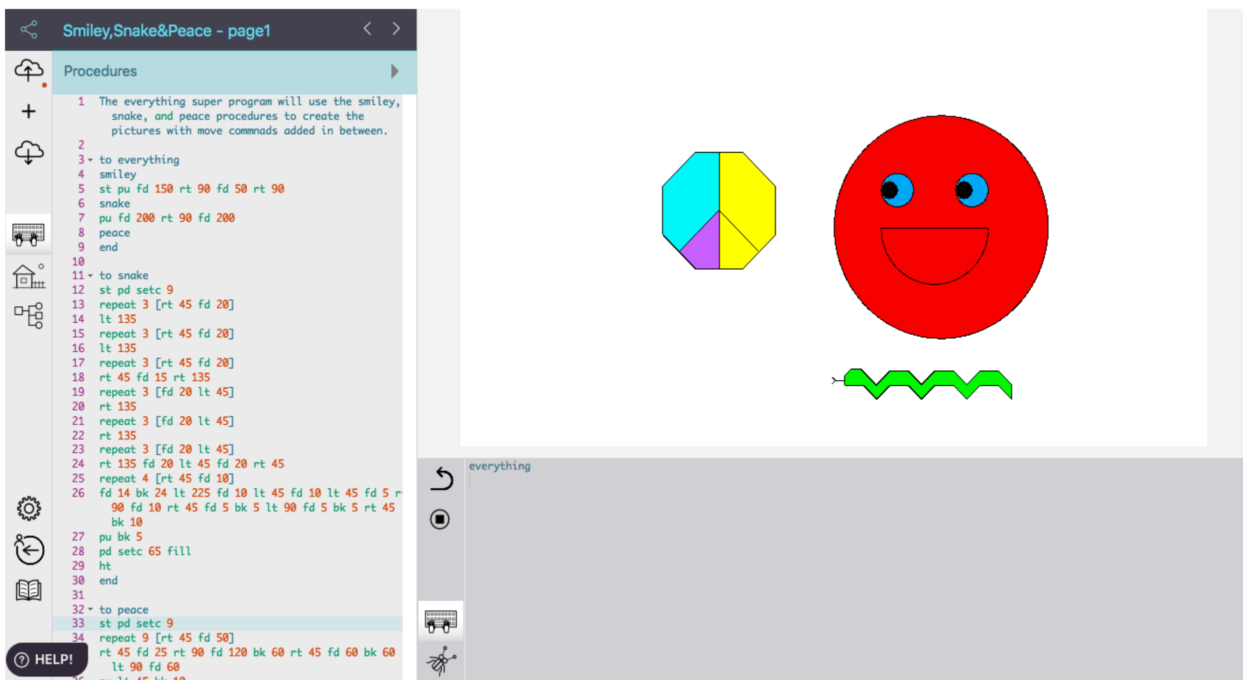


Figure 9. A student modular program with smiley, snake, and peace subprocedures. . (Refer to appendix for program procedures.)

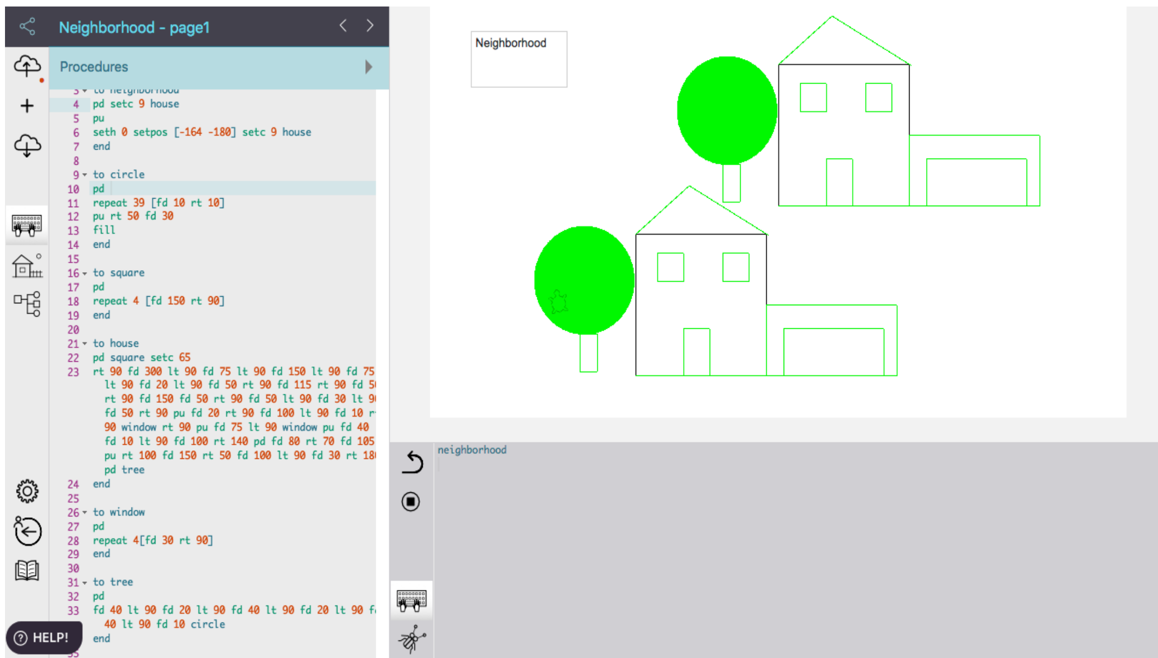


Figure 10. A student modular neighborhood program with house, tree, window, circle, and square subprocedures. (Refer to appendix for program procedures.)

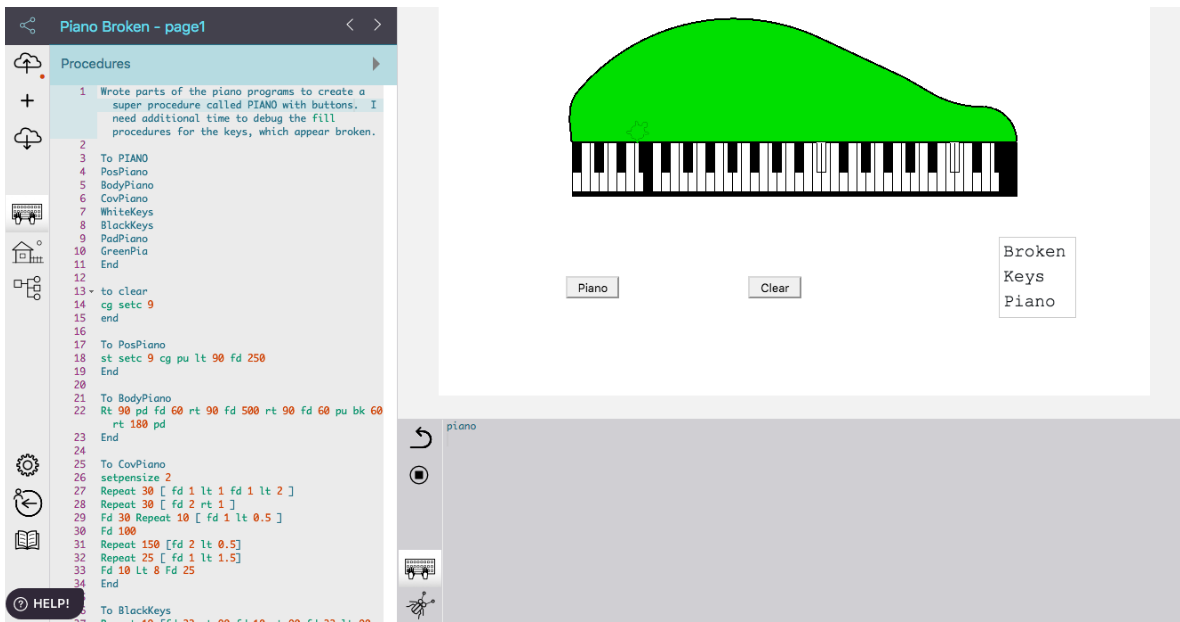


Figure 11. A student modular broken key piano program with various position piano part subprocedures. . (Refer to appendix for program procedures.)

Lynx Program Project – Modular Procedures

Turtle Activity 1

Create or use Logo procedures from previous projects to develop a modular program. Type the name of the program in the Command Center to test and run effectively without any error messages.

Turtle Activity 2
Select from the Appendix a coding project Figure 8 (`clock`), 9 (`everything`), 10 (`neighborhood`), and/or 11 (`piano`) to copy and run in the Command Center. Debug and add program procedures to enhance the project.

Turtle Activity 3
Create two or more turtle procedures and use these names to create an additional super procedure. Type the name of the super procedure in the Command Center to test and run properly without any error messages.

Simple Logo Recursion

A procedure that calls itself as a subprocedure in its final line is using **recursion** and is called a **recursive procedure**. You already have seen that in a modular program, a procedure can include calls to any other procedures defined in the project's Procedures Pane on the left side of the page. This means that **superprocedures** can call **subprocedures** defined in the project's Procedures Pane, for example, the superprocedure `house` calls the subprocedures `triangle` and `square` programs. Not only is it possible to call other procedures in a program, it is also possible to call the procedure *itself*. When this happens the program indefinitely repeats, as in this example of a recursive procedure:

```
to TRIANGLE                    Naming line
repeat 3 [fd 40 rt 120]
rt 45
TRIANGLE                       Name called again
end
```

In the above example, the turtle draws a triangle (the `repeat` instruction), turns right 45 degrees and then starts the triangle procedure again, drawing a triangle, then turning right 45 degrees, then starting the triangle procedure again, and so on. The program continues to call triangle until you press the Stop icon located just to the left of the Command Center.

Study the following Logo procedure examples:

```
to box
repeat 4 [fd 75 rt 90]
end
to frame
repeat 4 [box rt 90]
end
to hex
repeat 6 [fd 50 rt 60]
rt 10
hex
end
to figure
repeat 4 [fd 20 rt 90] fd 30
figure
end
```

Which procedures are modular and which show recursion? How do they work differently? The `box` program is a simple program while the `frame` program is modular. The `hex` and `figure` programs are recursive.

Student developing modular recursive program procedures can result in planned or unanticipated graphic outcomes. Some examples of student projects are a TX2 and shapes superprocedures (Figure 12 and 13).

Turtle Hint!
Suggest students follow these steps in developing a modular recursive program:

1. Write the program subprocedures (in other words, one or more separate procedures).
2. Develop the superprocedure using the names of the subprocedures in the program.
3. Add the superprocedure name at the bottom of the procedure, just before the end line.
4. Test the modular recursive procedure by typing the procedure name (for example, `TX2` or `shapes`) in the Command Center to see if the program runs correctly.

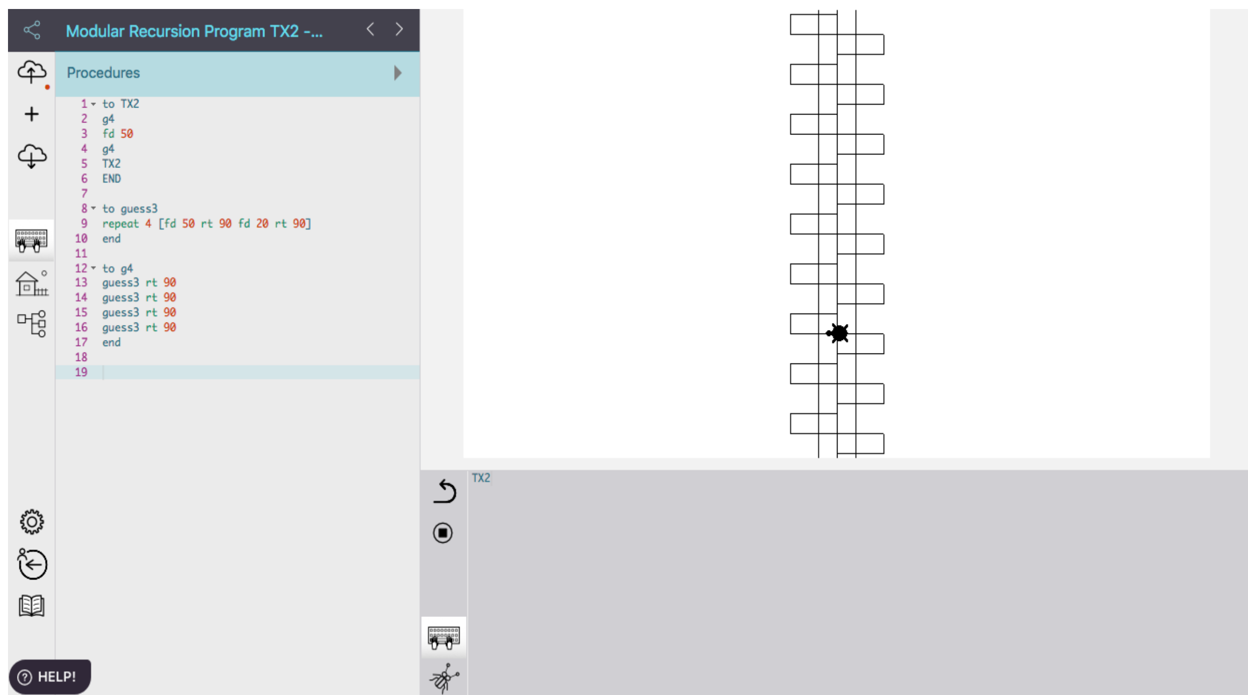


Figure 12. A student modular recursive named TX2 superprocedure calling g4 subprocedure. (Refer to appendix for program procedures.)

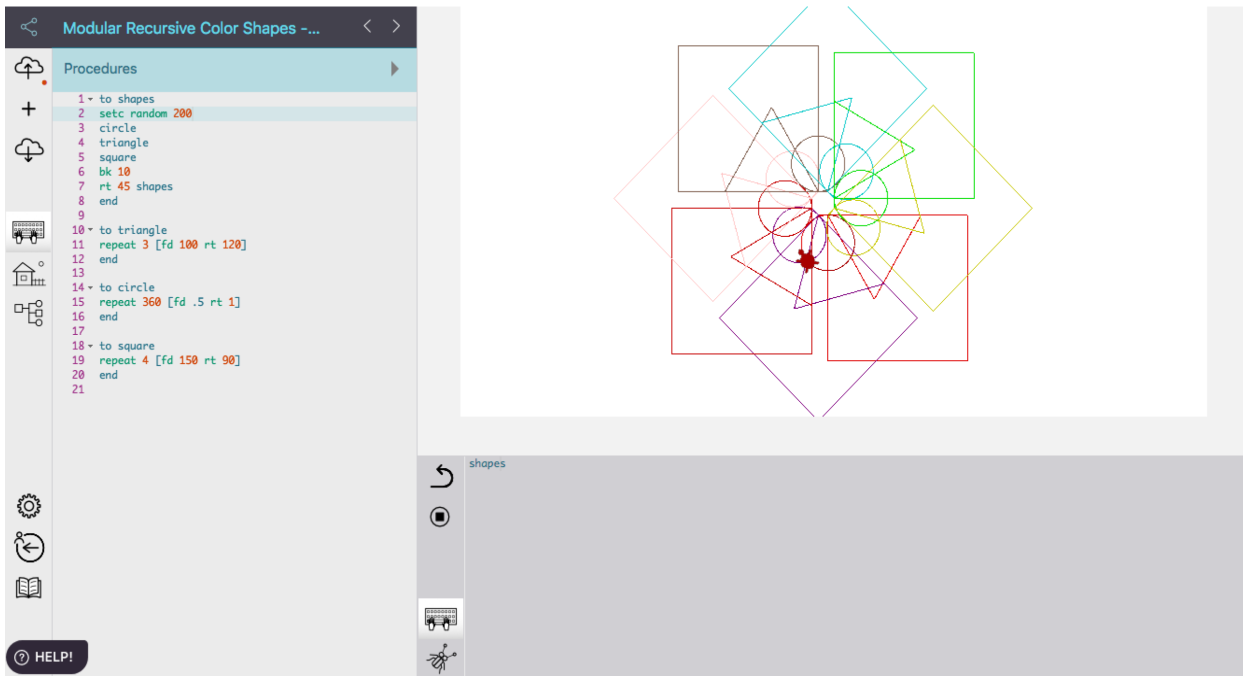


Figure 13. A student modular recursive superprocedure, including random color changing command, calling shape subprocedures. (Refer to appendix for program procedures.)

Lynx Program Project - Modular Recursive Procedures

Turtle Activity 1

Create or use Logo commands to write your own turtle graphics procedure and then add the procedure name again to show recursion. Type the name of the procedure in the Command Center to test and run properly without any error messages.

Turtle Activity 2

Select from the Appendix a coding project Figure 12 (`TX2`), and/or 13 (`shapes`) to copy and run in the Command Center. Debug and add program procedures to enhance the project.

Turtle Activity 3

Write a modular recursive program. Type the name of the superprocedure in the Command Center to test and run properly the subprocedures without any error messages.

Assigning Variables in Logo Programming

Naming things is an important function within Logo. Naming happens when teaching Logo new words to be used as procedures. To explain how variables work begin with the following procedure to draw a square with the pen down (`pd`):

```
to SQUARE
repeat 4 [fd 50 rt 90]
end
```

Type SQUARE in the Command Center and the turtle draws a square with sides of 50 units. You can create more powerful and flexible procedures by using variables. A variable is a name that stands for some value. To show a name is a variable and not, for example, a procedure name, you need to type a colon (:) in front of it. To show a procedure is using variables, the variable name is typed next to the procedure name in the naming line.

By using a colon and a variable name, for example (`:SIDE` or `:S`) with the `square` procedure, you can draw squares of different sizes using just one procedure. For example:

```
to SQUARE :SIDE          or        to SQUARE :S
repeat 4[fd :SIDE Rt 90]           repeat 4 [fd :S rt 90]
end                                end
```

Now the `SQUARE` procedure needs an input, just like some built-in Logo commands (such as `bk` and `lt`) need inputs when you use them. Type `SQUARE 100` and `SQUARE 25` in the Command Center. What happens? Try using another number after `SQUARE` and see what happens. The same `SQUARE` procedure draws all these different sized squares.

When creating a procedure with a variable, remember to use a variable name (for example, SIDE or S) after the colon in the procedure naming line and at the appropriate place in the instruction line(s). It is Logo tradition to pronounce the colon as 'dots'. For example, `:SIDE` is pronounced as 'dots SIDE'. The dots used in `:SIDE` mean 'the value associated with the name SIDE.' Remember to *not* put a space between the dots and the variable name in your program procedure.

For example, to create a triangle of different sizes, type a procedure like this in the Procedures Page:

```
to Triangle :S
repeat 3[fd :S rt 90]
end
```

By typing `triangle` *number* in the Command Center, different size triangles can be created (for example, `Triangle 10` or `Triangle 48`). As seen in the `Triangle` procedure example, the variable is added on the naming line and in the appropriate instruction lines of the procedure. An important rule to remember when using variables in Logo is:

The variable name must be shown in both the procedure naming line and the command line in the procedure that uses the variable.

More than one variable name can be assigned in Logo procedures. Each variable name must be preceded by its own set of dots. For example, the following two-input variable procedure can be used to draw rectangles of different sizes and shapes:

```
to RECTANGLE :HEIGHT :LENGTH
  fd :HEIGHT
  rt 90
  fd :LENGTH
  rt 90
  fd :HEIGHT
  rt 90
  fd :LENGTH
  rt 90
  end
```

If `RECTANGLE 100 10` were typed in the command center, the turtle would make a long narrow rectangle. `RECTANGLE 100 100` would result in a square rectangle with equal sides.

Variables written in Logo procedures, as shown above, are local variables. A **local** variable is a variable whose value is in memory only while a procedure is running. This means the program will run if this procedure name is typed in the Command Center and the procedure is written in the project's Procedures Page. Stated another way, the program will run in this project *only*.

Students will develop a variable program and use this with other program procedures to create turtle graphics. Used in this manner they are creating variable modular programming procedures. An example of a modular program with variables is a shoe coding project (Figure 14). Variables are used in the procedures for changing the shoe base, shoe color, and background.

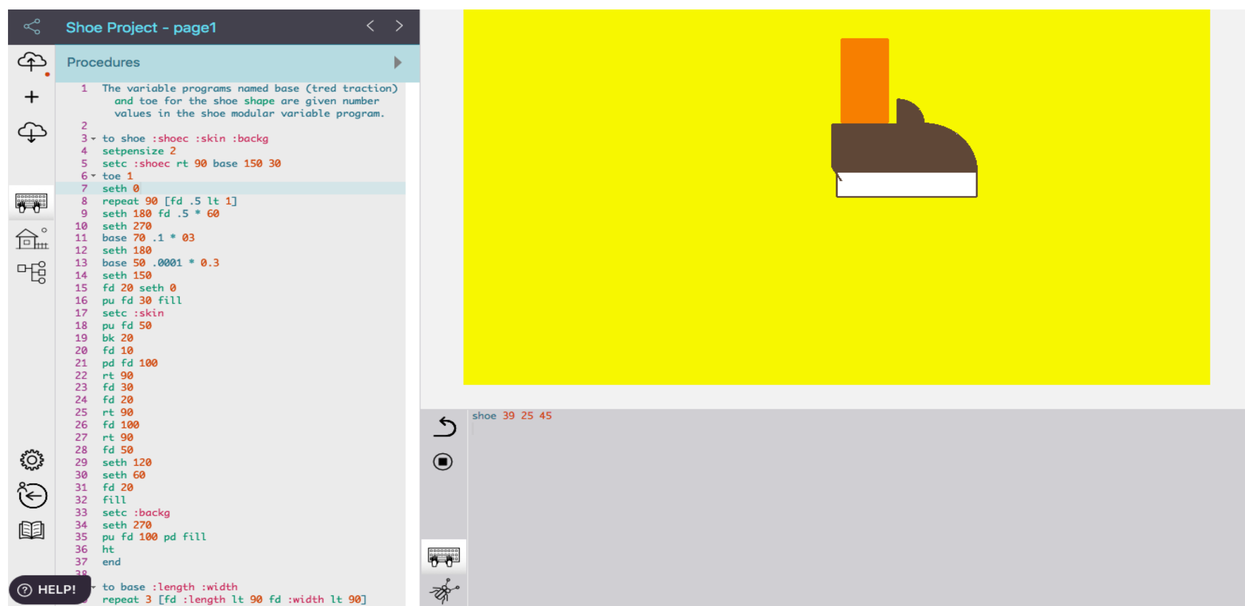


Figure 14. A student modular variable program procedures creating different shoe colors. (Refer to appendix for program procedures.)

You can create a modular variable program and then assign values to the commands in the program procedures. For example, a solar eclipse variable modular program with assigned variable values or numbers used in the program procedures was developed by a student (Figure 14). A circle variable program (`to circle: size`) was written to create the solar planet shapes with an assigned value of `circle 2` in the blackcircle subprocedure. In the solareclipse super program another circle assigned value was added in coding procedure (`circle 2`). Note the program procedures provides text after the semicolons to explain the coding project. Refer to these student modular variable programs for project ideas (Figures 15-18).

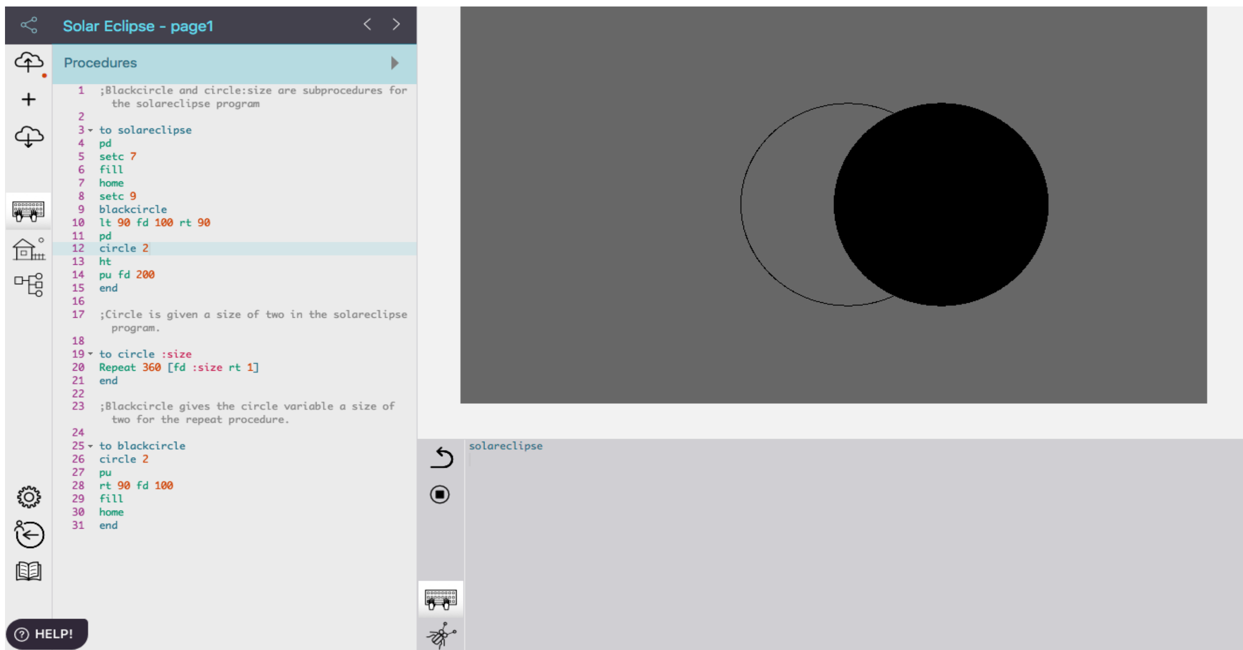


Figure 15. A student solar eclipse variable program with subprocedures. (Refer to appendix for program procedures.)

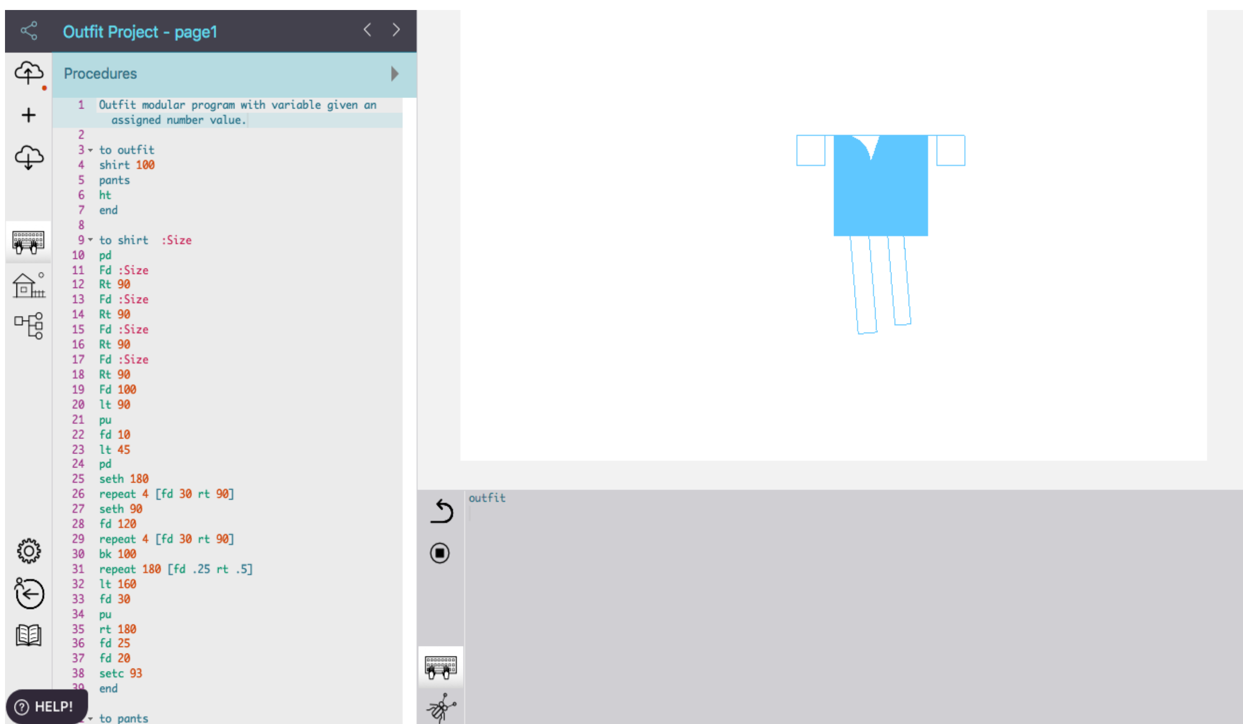


Figure 16. A student modular outfit procedures with assigned variable shirt values. (Refer to appendix for program procedures.)

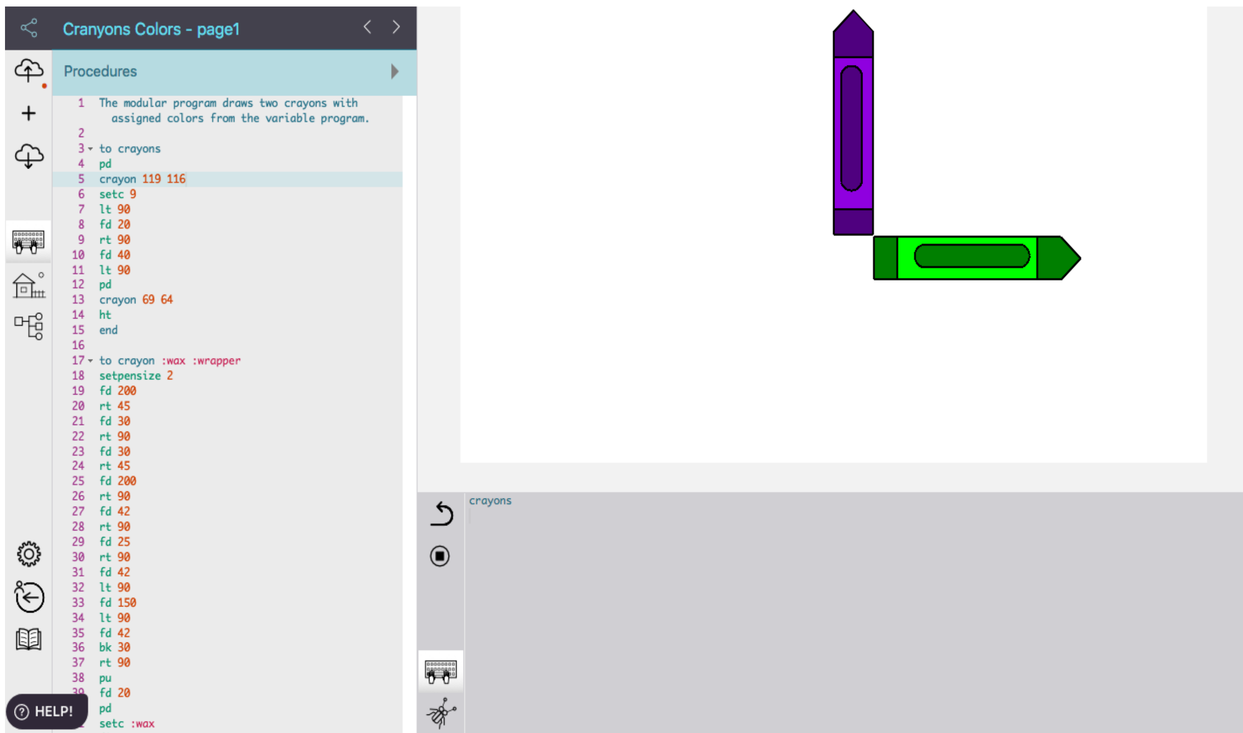


Figure 17. A student modular crayons procedures with assigned variable values. (Refer to appendix for program procedures.)

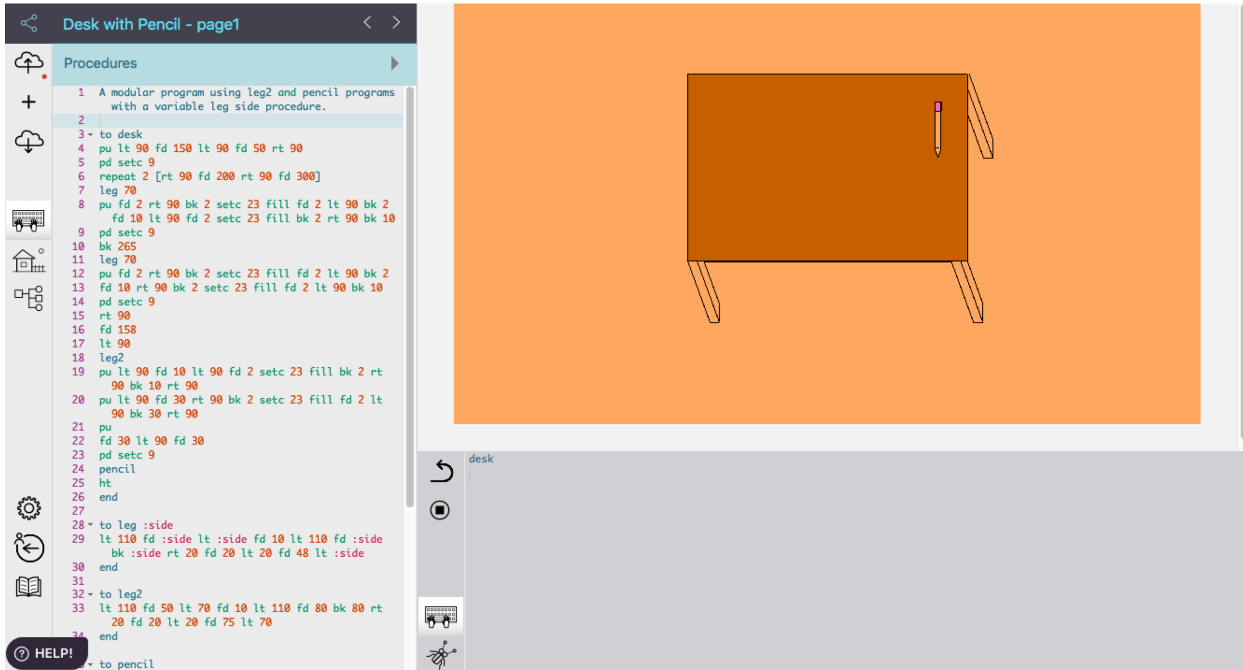


Figure 18. A student modular variable desk program using assigned variable values. (Refer to appendix for program procedures.)

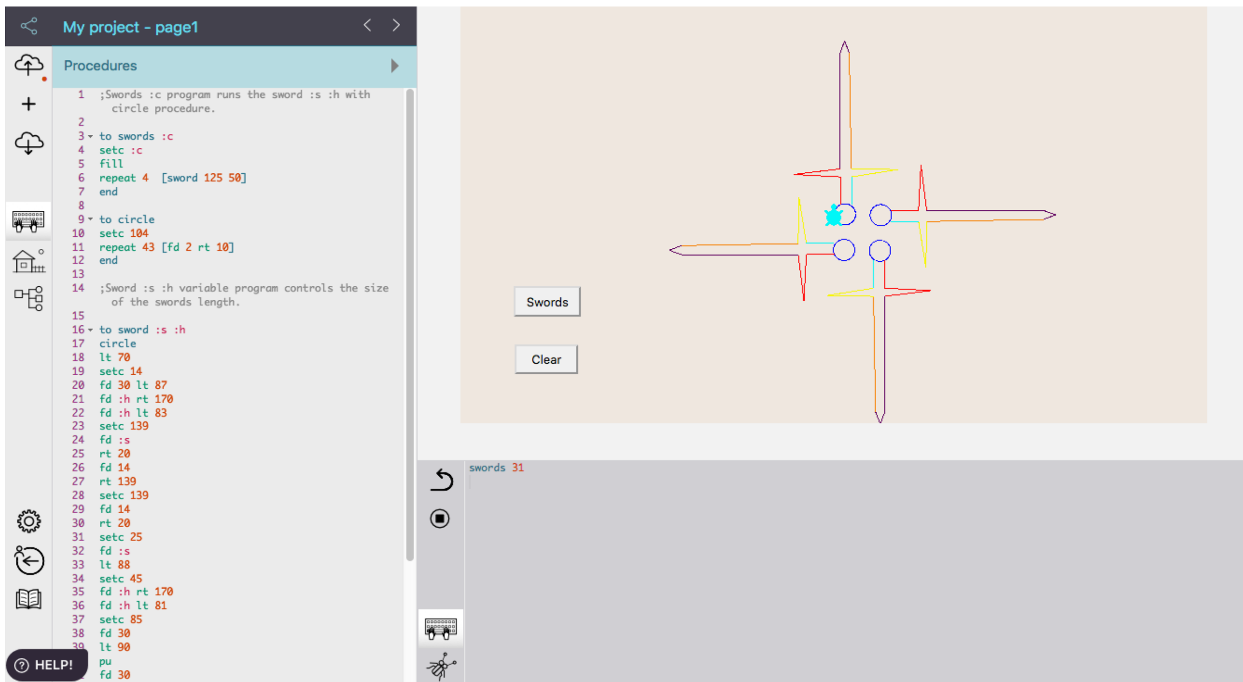


Figure 19. A student variable project clearing and creating medieval colored swords with buttons. (Refer to appendix for program procedures.)

Turtle Hint!

Suggest students follow these steps when developing a variable modular procedure:

- Write the program subprocedures (in other words, one or more separate turtle programs) first testing if the modular program works without added variables.
- Next, plan to add variable procedures deciding what parts of the program to allow for changing or program flexibility.
- Develop the superprocedure and make sure each subprocedure includes a value for each of the the defined variable names (for example, `Tri 25`).
- Add the superprocedure variable name(s) after the procedure name in the naming line and in the appropriate command lines in the procedure.
- Test the variable modular program by typing the procedure name and variable value(s)
- (for example, `House 67`) in the Command Center to see if the program runs correctly.

An alternative plan is to use similar variable names for the super- and subprocedures. For examples, study the modular superprocedure `Design` that uses the `Poly` subprocedure with variables:

```
to Design :R :D                             to Poly :R :D
repeat 5 [Poly :R :D Fd :D Lt 72]           repeat :R [fd :D rt 360/:R]
end                                         end
```

Lynx Variable Program Project

Turtle Activity 1
Create a turtle program and include at least one variable in the procedure. Type the name of the procedure, and a value for the variable, in the Command Center to test and run properly without any error messages.

Turtle Activity 2
Type the following variable procedures and test in the Command Center to see the various sizes of the objects. Modify and change the procedures to create your own idea. Which one is a modular variable superprocedure?

```
to MyBox :Size
fd :Size
rt 90
fd :Size
rt 90
fd :Size
rt 90
fd :Size
end
```

```
to House :Size
MyBox :Size
fd :Size Rt 30
tri :Size
end
```

```
to Tri: Size
repeat 3 [fd :Size rt 120]
end
```

Turtle Activity 3
Select from the Appendix a coding project Figure 7 (`laptop`) Figure 14 (`shoe`), 15 (`solareclipse`), 16 (`outfit`), 17 (`crayons`), 18 (`desk`), and/or 19 (`swords`) to copy and run in the Command Center. Debug and add program procedures to enhance the project.

Turtle Activity 4
Write a variable modular procedure with more than one variable. Type the name of the superprocedure in the Command Center to test and effectively run the program subprocedures without any error messages.

Recursive Variable Modular Procedures in Logo Programming

We have discussed that a repeating program that defines a procedure that includes a call to itself in the final program line is called **recursion**. You already known that the steps of a procedure can include calls to any other procedures in modular programs. This means that **superprocedures** can call **subprocedures** that are defined in the Procedures Pane on the side of a page, like when `triangle` and `square` procedures are called (or used) in the superprocedure `house`. Since it is possible to call other procedures in a program it is also possible to call a procedure line itself (in other words, the name used in the naming line of the procedure). When this happens the program indefinitely repeats. The following is an example of a repeating program:

```
Naming Line          to TRIANGLE
                     repeat 3 [fd 40 rt 120]
                     rt 45
Name Called Again    TRIANGLE
                     End
```

In the above example, the turtle draws a triangle, turns right 45 degrees, and then draws another triangle. The procedure continues to call `triangle` (in other words, make another `triangle`, turn `right 45` degrees, and call `TRIANGLE`) until the user stops the recursion or repeating action.

What if we decided to continue the repeating recursive action and add variables to the recursive program? An example of a recursive program that adds variables to the `TRIANGLE` procedure above is:

```
to TRIANGLE :SIZE
repeat 3 [fd :SIZE rt 120]
TRIANGLE :SIZE + 5
end
```

To run this program, you need to type a number after `TRIANGLE` in the Command Center. Remember, this triangle recursive program has dots and a variable name called `SIZE (:SIZE)`. Because a variable is used in this procedure, a number must be typed after the procedure name in the Command Center (for example, `TRIANGLE 15`). The procedure then uses the number assigned to size after the fd command and, because `TRIANGLE :SIZE` is called in the last line of the procedure, it repeats (recurses), making the triangle over and over again. Adding the number 5 after `:SIZE` means the value of `:SIZE` increases each time `TRIANGLE` repeats, causing the size of the triangle to increase (the triangle's sides become 5 units or turtle steps larger). Once you type the procedure name and a number for the variable size (for example, `TRIANGLE 15`) in the Command Center, the graphic is drawn repeatedly on the screen.

A recursive program can be stopped using an if statement. The `if` statement, called a conditional expression or statement, can be added to the program. The `if` statement has the form:

`if` [this is true] [then do this action].

The action is always a list of one or more instructions enclosed in square brackets. For example, to stop the `TRIANGLE` program from recursing on the screen, an **`if`** statement can be written in the program:

```
to TRIANGLE :SIZE
if :SIZE > 50 [stop]
repeat 3 [fd :SIZE rt 120]
TRIANGLE :SIZE + 5
end
```

In the above `TRIANGLE` program, Logo will draw the graphic over and over again until the value of `SIZE` equals 50. For example, if you type `TRIANGLE 5` in the Command Center, each time the procedure runs the value of `:SIZE` increases (5 + 5 = 10, 10 + 5= 15, 15 + 5 = 20, and so on) and the if statement checks the value to see if it is greater than 50. If it is, the procedure (and drawing) stops.

In addition to the variable (`:SIZE`), the predicate (>), the number (50) limiting the size of triangle, and the stop command in square brackets are shown as part of the if statement. The first input to if must always be a condition that tests and reports if something is either true or false. The statement usually has several pieces – a value, a predicate, and another value. You can use the following predicates: equal (=), less than (<), or greater than (>). Remember to put spaces between predicate symbols and both the first and second values that you are comparing. If the condition (in this example, `:SIZE > 50`) is true, Logo runs the list of instructions inside the

square brackets, in this example, `Stop`. If the condition is false, Logo ignores the instruction and goes to the next line in the procedure.

Study the following recursive variable tower program using an `if` statement.

```
to TOWER :SIZE
if :SIZE < 3 [stop]
SQUARE :SIZE
fd :SIZE
TOWER :SIZE * 0.6
end
```

Subprocedure used in TOWER:

```
to SQUARE :SIZE
repeat 4[Fd :SIZE rt 90]
end
```

The `TOWER` program is a variable program with an if condition statement. It is also a recursive program causing the size of the tower to change. In `TOWER :SIZE * 0.6` the variable size used as input to `TOWER` in the Command Center is multiplied by the decimal number, creating a new value for `:SIZE`. The `TOWER` program is also modular because of the `SQUARE :SIZE` subprocedure.

Can you explain what each line of the recursive variable modular program does after you type it in the Procedures Pane on the side of the page? Run and test the program to see if you can make different sized towers. Did you remember to include a subprocedure program `SQUARE :SIZE` in order to have the `TOWER` program work? What happens to the size of the towers when the procedure is tested in the command center?

Lynx Modular Variable (Recursion) Project

Turtle Activity 1
Type the following variable programs and test them in the Command Center to view the results. Modify and change the programs to create your own ideas. Which programs are simple variable, variable recursive, or show modular variable recursion?

```
to Spiral :S
repeat 100 [fd :S rt 90 make "S :S + 5]
end
to Hex :Num
setc 5 setbg 1
repeat 6 [fd :Num rt 60]
lt 150
Hex :Num
End
to Growsquare
```

Superprocedure for Square `:Size` program

```
Square 10
Square 20
Square 30
Square 40
Square 50
Square 60
Square 70
Square 80
Square 90
Square 100
Square 110
```

```
Square 120
end
to Growsquares :Size   Superprocedure for Square :Size program
if :Size > 90 [stop]
Square :Size
Growsquares :Size + 10
end
To Square :Size        Subprocedure for GrowSquare and
Repeat 4[Fd :Size Rt 90]   Growsquares :Size
End
```

Turtle Activity 2
Apply Logo procedures and primitives to write your own turtle modular variable recursive program. Test your program to see if it works in the Command Center without any error messages.

Animating Turtle Shapes

By applying what you have learned about basic turtle coding and drawing shapes, you are ready to create animated or moving graphics. Consider the following program to show an animated moving dog:

```
to Rundog
pu seth 90
repeat 50 [setsh 30 wait 2 setsh 31 wait 2 fd 10]
end
```

In this program you can see the pu command is used so the turtle will not leave a trail. The turtle has then been told to wear two costumes (in this case, `setsh 30` and `setsh 31`) and to repeat changing these shapes 50 times. The command wait causes Logo to pause for 2/10th of a second (if you use `wait 10`, Logo pauses for about one second). Type this procedure in the Procedures Pane on the left side of the page.

Before running the procedure you will need to make the animation look like a dog, instead of a moving turtle. First, you need to find the dogs from the "+" symbol on the toolbar to open the window. Scroll to the Sample Clipart window to view the side window selecting Animation. Find the two dog shapes (numbers 30 and 31), click on the dog shape to view the hand image on a hatched turtle on the page. Select the other dog shape and click on the turtle again. Select the keyboard image on the toolbar to view the Procedures side window displaying the Rundog program. Finally, you can type "rundog" in the command center to see if the dog moves across the screen (refer to Figure 20) and stop the procedure by pressing the stop button icon. Remember `setsh 0` returns the turtle to its original shape, a turtle. You can import your own shapes to the shapes page. Refer to *Getting Started with Lynx* pages 20-22 for further instructions on using clipart as turtle shapes and adding your own clipart. This PDF is in the Help Section>User Guides.

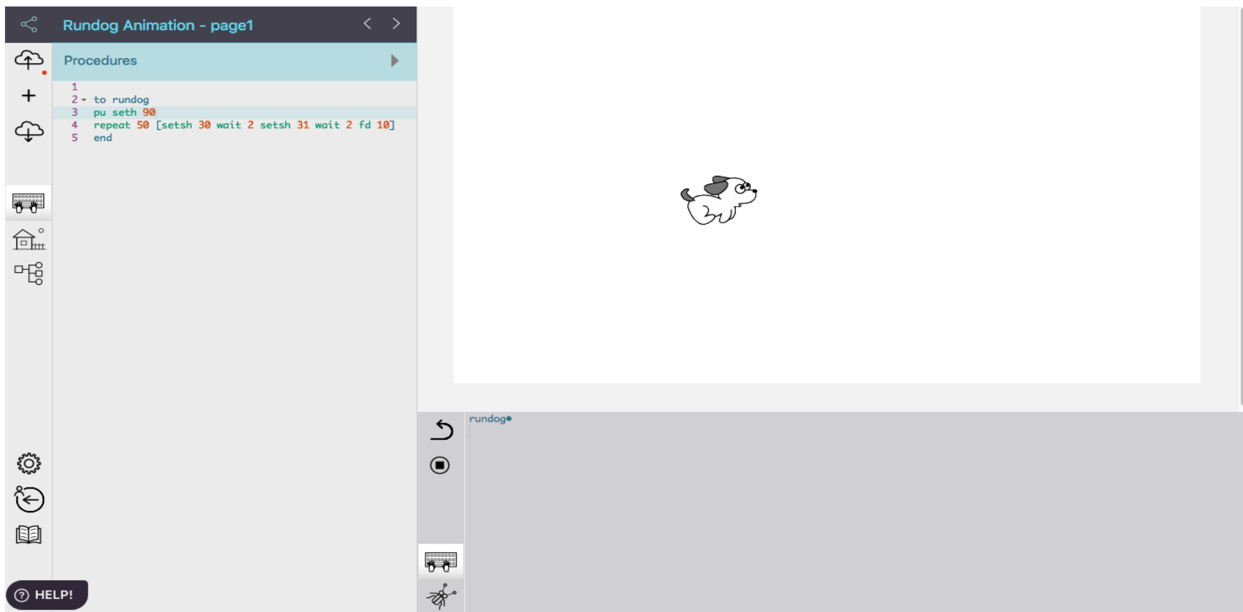


Figure 20. View of the Rundog program entered in the Lynx program.

Adding a Button and Slider for Animation of Shapes

To animate more than one shape, for example, two running Lynx turtle shapes you can hatch two turtles from the plus “+” symbol and then write the following program in the procedures window:

```
to Lynxrace
tto [t1 t2] seth 90
repeat 100 [tto [t1 t2] setsh 1 fd random 20 wait 1 setsh 2
fd random 20 wait 2 setsh 3 fd random 20 wait 1 setsh 4 fd
random 20 wait 2]
end
```

Type `Lynxrace` in the command center to run the program. Do the Lynx run together or at different speeds? Refer to Figure 21.

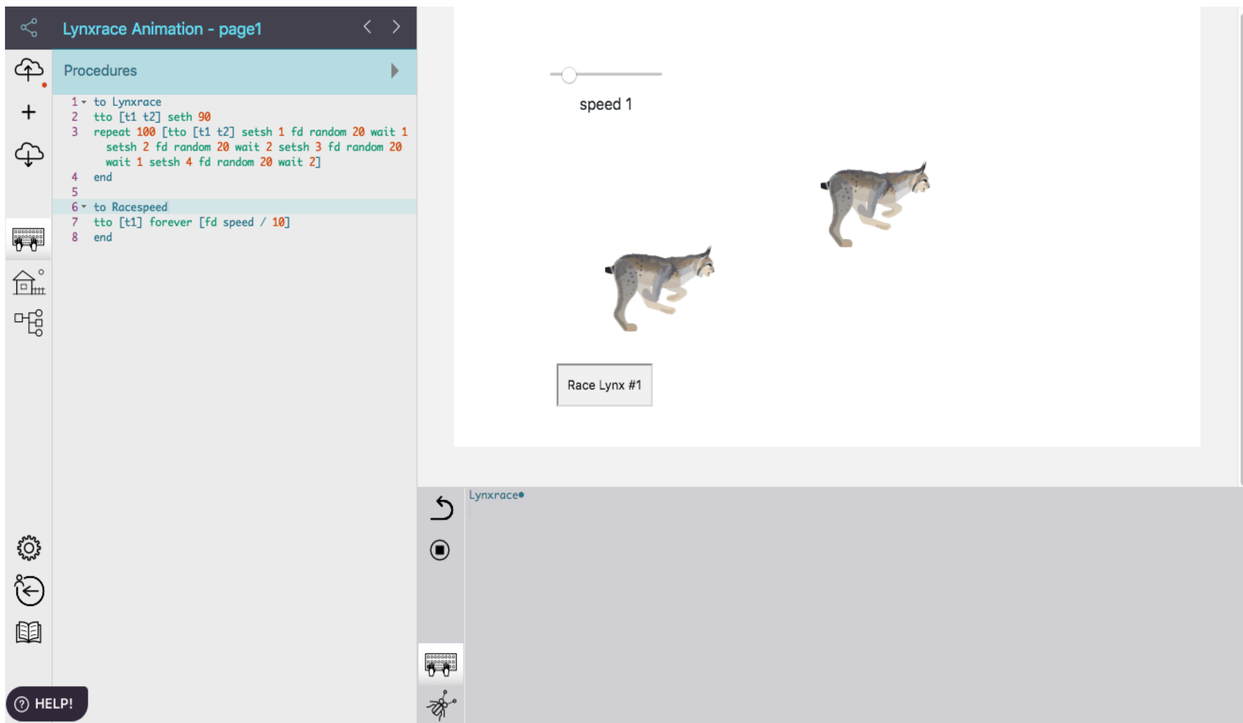


Figure 21. An example of a Lynx race animation using a control buttom to adjust the speed of one Lynx.

By adding a button and slider tool you can vary the speed of a Lynx. Add a button to the page by selecting from the plus "+" symbol. Label the button with a name (for example, Race Lynx #1) and then type or select the program name (for example, `Racespeed`) in the On click space Refer to Figure 22. To vary the speed of an object, like the Lynx, choose slider from the plus "+" icon. Open the slider (Cntrl-Command or right click) to Name the slider speed (Refer to Figure 23). Set the Min and Max values from 0 to 10 with a Value of 0 and then click the Apply button. Write the following `Racespeed` program as follows in the procedure window:

```
to Racespeed
tto [t1] forever [fd speed / 10]
end
```

Type `Racespeed` in the command center and find out which turtle shape Lynx t1 or t2 can you vary the speed by clicking the button and moving the slider?

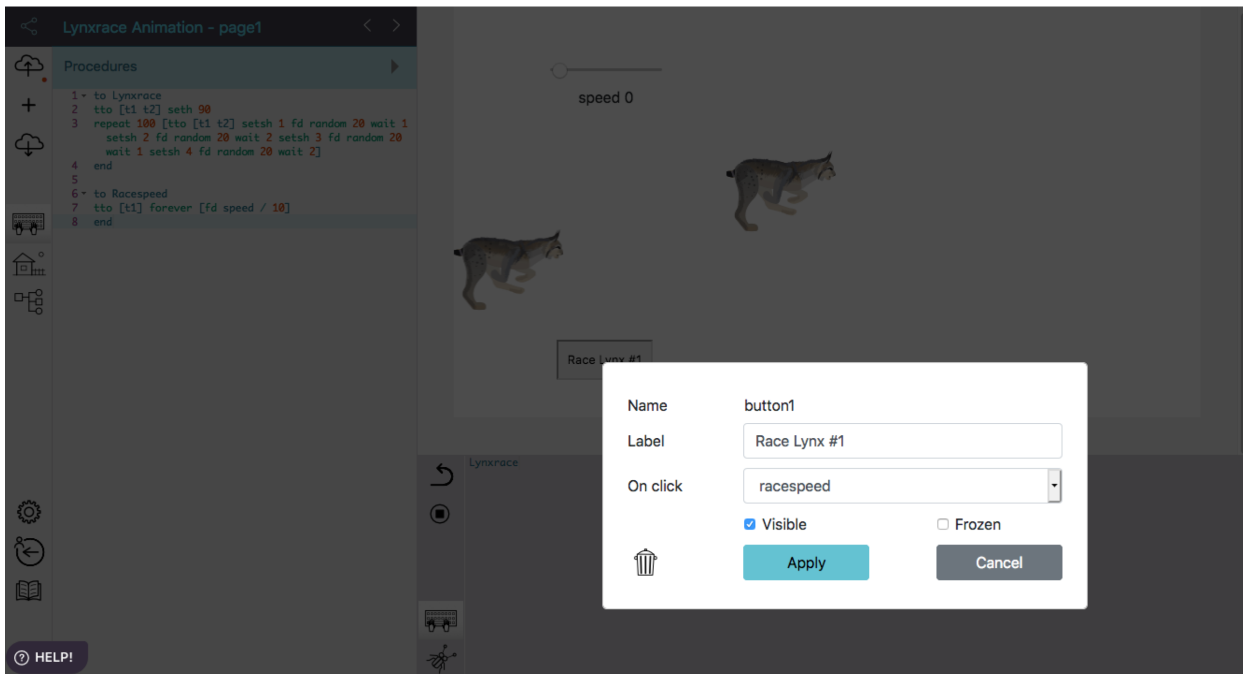


Figure 22. View of window for creating a button for the Lynx race.

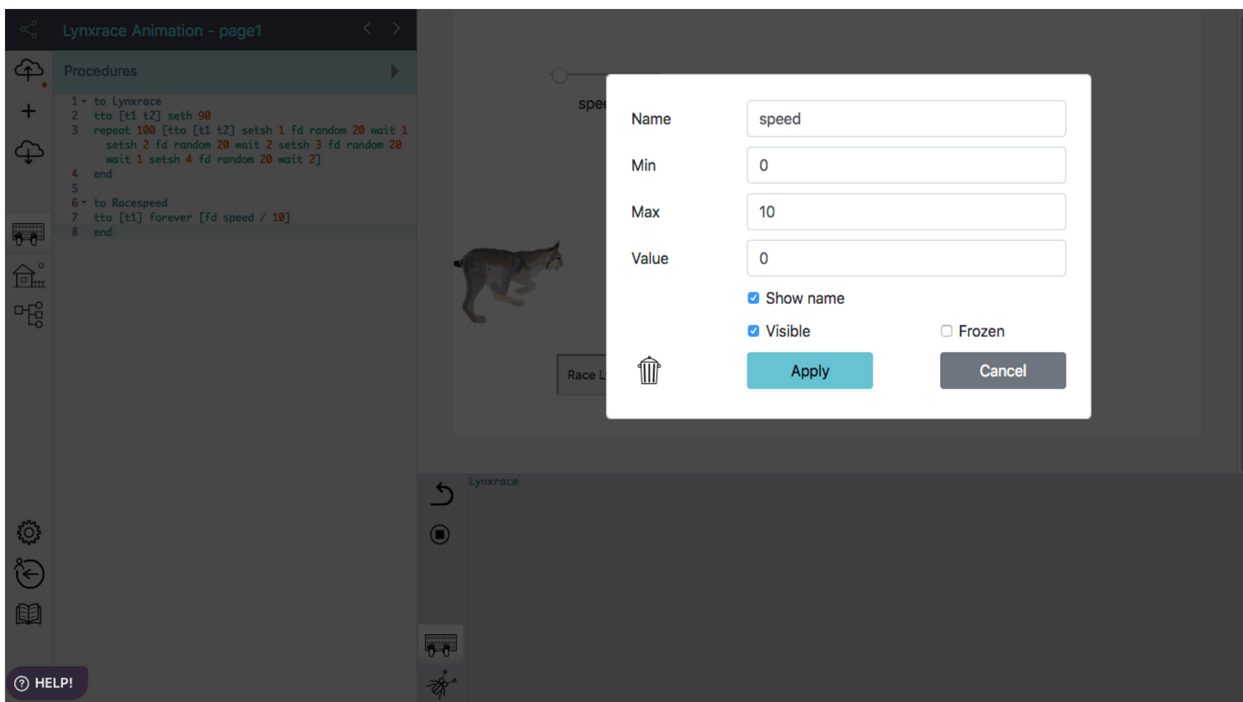


Figure 23. View of the tool window showing the slider name speed with minimum and maximum values.

Animation Procedures with Varying Shape Speed and Added Background

Another option is to write an animated coding procedure moving turtle shapes at different rates of speed in front of a background scene. Begin by typing the `Lynxchase` program in the procedures window:

```
to Lynxchase
tto [t1] seth 90 forever [setsh 1 fd random 50 wait 2 setsh 2 fd
random 50 wait 2]
tto [t2] seth 90 forever [setsh 1 fd random 50 wait 2 setsh 2 fd
random 50 wait 2]
end
```

Select from the plus "+" symbol to hatch two turtles (`t1` and `t2`). Open the Sample Clipart window to find the four Lynx shapes (numbers 1 to 4). Click on each Lynx shape on to t1 and t2. Remember to view the hand image on a hatched turtle on the page for clip art placement. Select all four Lynx shapes by repeatedly clicking on the turtle. Select the keyboard image on the toolbar to view the Procedures side window displaying the `Lynxchase` program. Finally, you can type `Lynxchase` in the command center to see if the two Lynx move across the screen Do the Lynx run at different speeds? Which commands vary the speed of the Lynx?

A background scene can be selected after running the program successfully. To include a background click on the "+" symbol on the tool bar and open the window. Select the sample Clip art side window Background or other objects like Buildings, Nature, or People. You will be giving the background shape to a turtle and stamping this into place. Refer to *Getting Started with Lynx* page 24 for further instructions on adding a background scene to your project. Figure 24 shows the `Lynxchase` program with a background scene.

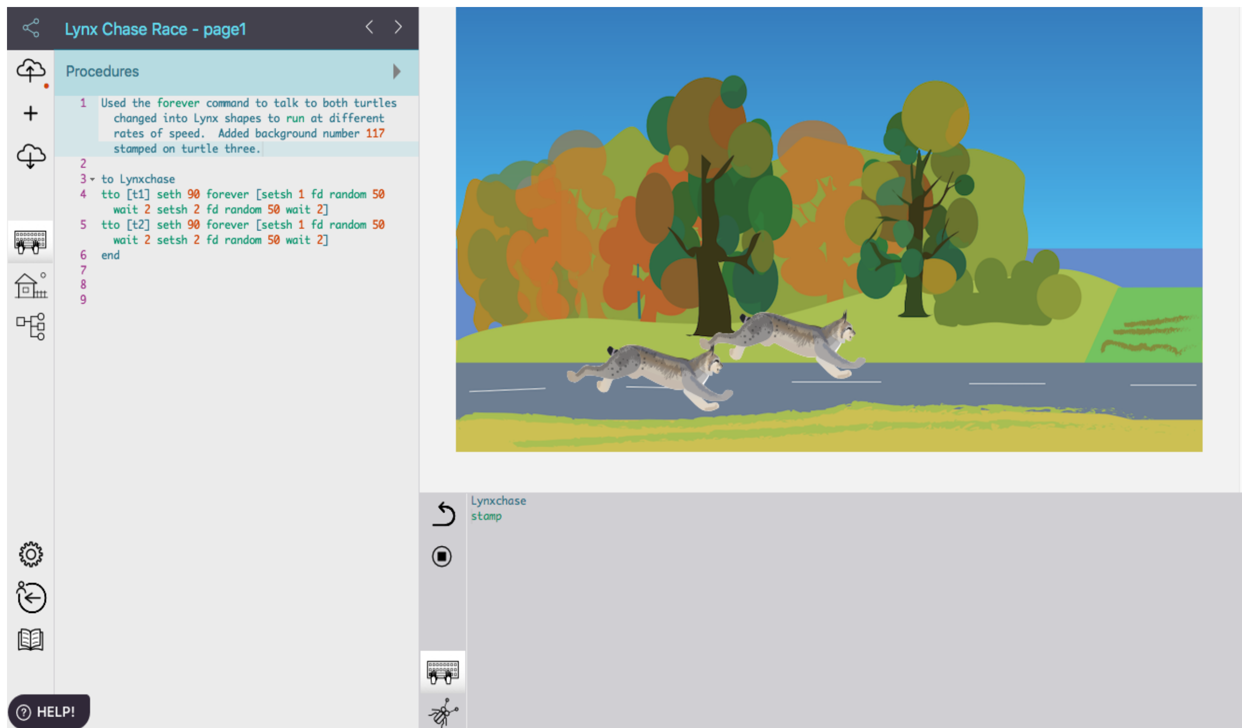


Figure 24. Example of two Lynx racing at various speeds with the addition of a background shape.

Additional Features for Project Development

Adding Pages

You may want to have more than one page to create presentations of all of your projects including animation displays. A cover page may be written to provide a title page of the project or directions for viewing and use. Remember to click the left and right arrows (< >) located a the top of the Procedures Pane to switch pages after chosing Page in the "+" menu. You can also add buttons to advance pages by creating a button link to a page using the following example program procedure:

```
to Page2
page 2
end
to Page1
page1
end
```

Further instructions about adding and changing pages can be found at *Getting Started with Lynx* on page 18.

Adding Sound and Music

Adding sound and music to your animation scene can be downloaded and imported into the project. Begin by chosing Sound in the "+" menu for the Import sound . . . dialog box to appear. Select (button) a sound file to add the URL link. Click the Create button to import the sound link to the project page work area. A musical note icon will appear with an attached name, which can be changed with a right-click or Control-Command click on the icon. *Getting Started with Lynx* on page 17 provides additional music ideas along with the BBC Sound effects link at http://bbcsfx.acropolis.org.uk/.

A Clickable and Detectable Turtle to Control Movement

A clickable turtle allows the user to click on a turtle shape and have the turtle perform the command(s) written in a program. Begin by typing a program procedure to control the movement of a turtle, for example:

```
to command
pd repeat 4 [fd 100 rt 90]
rt 45
end
```

Next right click on a turtle to view the Name window and select the program name (command) in the On click field. Click on the Apply button to close the window. Click on the turtle several times. What did you make?

When opening the turtle's Name window you can also control the turtle movement for other objects using touch and color. Refer to *Getting Started with Lynx* pages 25-28 for instructions and procedures for controlling the turtle.

Turtle Hint!
Turtle control reminder to right click on the turtle to find the turtle name (for example, `t1` or `t2`). Change the name of a turtle with the window display open to another turtle number or Name like "Tom. The name must be 1 word. You can also change the Xcor and Ycor positions of the turtle.

Lynx Animation Program Project

Turtle Activity 1
Type the `Rundog` animation program procedures and hatch a turtle for placement of the Lynx dog shapes. Add a background scene for the animated graphic.

Turtle Activity 2
Run the `Lynxrace` program and add a button with a slider to vary the speed of one Lynx. Alternately run the `Lynxrace` program with a selected added background.

Turtle Activity 3
Develop your animation program by adding sound, music, and pages including controlling the turtle on a click or to detect objects.

Turtle Activity 4
Create your own animated graphic using ideas from the Lynx at https://lynxcoding.club/. Select the Learner Mode button are review the following two programs for ideas:

- Terry Fox
- On Your Mark, Get Set . . .

Select the Advanced button to review the following program for ideas:

- Down the River

Going Further: Words and Lists in Logo Procedures

The Print Statement and Character String Changes
Logo provides operations for manipulating words. Words may be combined into longer words or broken into word parts. Words may be written as procedures and may serve as inputs or outputs. A string of characters (for example, letters of the alphabet) is called a word. Logo permits the user to manipulate and change sequences of words on a word-by-word basis. The following are some procedures for using primitives to control words.

To tell Logo you are typing a word, and that word is **not** a command, always place apostrophes before and after the character string (word). Alternately, a quotation mark before the character string can be used. For example, create a textbox on the page and try:

```
print ‘Turtle’ or print “Turtle
```

Logo will type:

```
Turtle
```

`Print` tells Logo to print the word Turtle. Print may be abbreviated pr when typed in the Command Center or in a procedure (for example, `pr "Turtle`). The quotation mark is only needed at the beginning of the word. If a quotation mark is placed at the end of a word, Logo thinks the quotation mark is part of the word:

```
print "Turtle"
Turtle"
```

With the `Print` statement it is possible to manipulate and change printed words. Logo provides the following operations for taking characters *out of words* using **first**:

`first`	Outputs the **first** character of a word used as input
`butfirst` (or `bf`)	Outputs a word containing all **but** the **first** character of a word used as input

Some examples using these operations include:

```
print first 'slow'
S
print butfirst 'slow'
low
print butfirst 's'
(blank line)
```

Logo provides the following operations for taking characters *out of words* using **last**:

`last`	Outputs the **last** character or a word used as input
`butlast` (or `bl`)	Outputs a word containing all **but** the **last** character of a word used as input

Some examples using these operations include:

```
print last butlast 'slow'
o
print butlast 's'
(blank line)
```

The instruction print `last butlast "slow` says to print the last letter of its input. In this case, its input is whatever `butlast "slow` outputs, which is all but the last character of the word `slow`. This means, `butlast` outputs `slo`. Since the last character in slo is o, Logo prints `o`.

The result of `print butfirst "s` and `print butlast "s` is a word containing no characters. A no-character word is called an empty word. An empty word can also be specified by typing a quotation mark followed by no characters:

```
print ' '
(blank line)
```

Another command, item, is similiar to the character operation commands. `Item` reports the element of a word or list that is in the specified position. For example:

```
print item 2 'slow'
l
```

For making a larger word from smaller ones, Logo provides the `word` operation. `Word` takes two words as input and combines them to form a single word:

```
print word 'base' 'ball'
baseball
```

A sequence of words is called a `list` in Logo. Use the list command to `print` two character strings (words), leaving a space between them.

```
print list 'base' 'ball'
base ball
```

In Logo, numbers are treated as words when used as input to any commands that change words. All word changing commands work on numbers, too.

```
print first 2468
2
print butfirst 2468
468
print word 3412 2935
34122935
```

Note that a quotation mark is *not* required when typing numbers after print commands.

Defining and Manipulating Words, Lists, and Sentences

Since Logo allows you change sequences of words on a word-by-word basis and numbers are treated as words, it may be helpful to discuss rules and procedures for writing words and lists. This includes procedures for writing sentences, which will be helpful in successfully running turtle programs and displaying outcomes. Understanding correct Logo terms will lead to successfully writing interactive programs using words and lists to show conversations.

A sequence of words is called a list. Yoder (1985) describes three kinds of objects which make up the Logo language: words, numbers, and lists. These three objects are summarized as follows:

1. Word - An ordered collection of characters. Often Logo words contain letters, numbers, periods and symbols (for example, &, >, and +)
2. Number - A special kind of word that does not need apostrophes or quotation marks
3. List - An ordered collection of words and/or lists
   (Yoder, 55, 1988).

A space usually separates two different Logo words. This means any name of a Logo program or procedure must be a single Logo word and cannot have a space in it (for example, MYHOUSE would work, but MY HOUSE would not). If a word does not have a quotation mark in front of it, Logo treats it as a primitive or procedure name – something to run. If you put a quotation mark in front of a word, Logo does not treat this as a command, but recognizes it as a word. For example, when `print "Hello` is typed in the Command Center, Logo recognizes that Hello is a word, not a command, and prints `Hello` in the textbox on the page. If `Print Hello` (without the quotation mark) is typed, and because Hello is not a built-in Logo word, Logo looks for a procedure named Hello. Since there is none, Logo responds with "*I don't know how to Hello.*"

Numbers are special kinds of words in Logo and don't require quotation marks. For example, `print 456` prints the number 456 in the textbox on the page. This is so Logo can use standard arithmetic notation when solving problems, for example,

```
print 45 + 67
```

or

```
pr 5678 / 23
```

Sometimes Logo users are unsure if quotations, brackets, parentheses, or colons should be used when working with words and lists. These questions sometimes arise when debugging Logo programs. As a rule, ask yourself the following question: "Do I want Logo to read a sequence of

words (in other words, a list) or group some part of an instruction here?" In addition, keep these punctuation rules in mind:

Symbol	Name	Use
`' '` or `"`	Apostrophes or Quotation mark	Used to designate a word, rather than a primitive or procedure name
`[ ]`	Square brackets	Used to indicate a list
`( )`	Parentheses	Used as grouping symbols, to clarify how inputs are evaluated
`:`	Colon or 'dots'	Used as grouping symbols, to clarify how inputs are evaluated

For example, the bracket may be used to define words or lists as follows:

Example	Description
`[Debug the turtle]`	A list of three words
`[678 a lsjlksa]`	A list of three words
`[Debug [the] turtle]`	A list of two words and one list (containing one word)
`[ [Debug now] [Debug turtle] [Debug the turtle] ]`	A list of three lists

Review the above examples of lists to get a better understanding of the bracket punctuation rule.

Parentheses are used to group symbols together. For example:

```
print (list 'hello' 'green' 'turtle')
```

List usually takes only two inputs. When using more than 2 inputs (for example, three as in the example above) you need to put parenthese around the command and all of its inputs. If you are using fewer inputs (for example, to change a word from a word to a list, which is sometimes necessary when using certain primitives), you need to put parenthesesseperating them around the command and all of its inputs.

```
print(23 + 56 + 98) / 4
```

In this example, the parentheses clarify the arithemetic operations, and are used just as they would be in paper and pencil arithmetic operations.

The colon (or 'dots') indicates a variable and means " the value associated with a name". For example, when assigning variables in Logo procedures, you would type a name with a colon (:) in front of it (`SQUARE :SIDES`). When running the procedure, the appropriate type of input (in this case, a number) is used as input, for example, `SQUARE 50`.

While gaining experience working with Logo procedures and debugging program errors, these rules of grammar and punctuation will become more clear. Remember to ask the question: "Do I want Logo to read a sequence of words (in other words, a list) or group some part of instructions for running a procedure?"

Further information, examples, and applications of these rules regarding Logo grammar and punctuation are discussed below. This will provide the Logo user additional skill to effectively write procedures using words and lists.

Words may be listed with separate spaces separating them and enclosed in square brackets. When words are in a list, they do not need a quotation mark and the square brackets are not printed. Extra spaces are ignored when typing lists of words in a bracket and result in the following textbox display:

```
print [The turtle makes lists.]
The turtle makes lists.
```

`First`, `last`, `butfirst`, and `butlast` commands operate on lists and words in a similiar manner. When used with lists, these operations pick out the first or last word of a list, rather than the first or last character of a word. Here are a few examples of operation commands and their resulting screen displays:

```
print first [The turtle makes lists.]
The
print first butfirst [The turtle makes lists.]
turtle
print butlast [The turtle makes lists.]
The turtle makes
print butfirst [The]
```
(blank line)

The last example produces no words and is called an empty list. An empty list may be typed into a program as `print[]`. Typing `print empty?` *word/list* reports *true* if a *word/list* is an empty word or an empty list; otherwise it reports *false*. Examples using `print empty?` are:

```
print empty? []
true
print empty? "
true
print empty? ‘ ‘
false
print empty? [0]
false
```

Even though Logo prints words and lists the same way, they are not considered equal. This is shown in the following example:

```
print ‘Turtle’                {list}
Turtle
print [Turtle]                {word}
Turtle
print ‘Turtle’ = [Turtle]     {rule proven}
false
```

Remember, a string of characters are called a word and a sequence of words are called a list. As a general rule a list in Logo is never considered equal to a word. Conversely, a word is not considered equal to a list that contains that single word. This word/list inequality exists even though Logo prints these in the same way. The `print Word?` *word/list* command reports true if an input is a word, false if it is not. Study these commands and screen displays:

```
print word? ‘shapes’
true
print word? [shapes]
false
```

```
print word? 123
true
```

As shown above, shapes is a word as long as it is not enclosed in square brackets. `Print word?` *word/list* also proves a list and a word are not the same thing. Remember, square brackets are used to define a sequence of words called a list. Finally, as the last `print word?` instruction shows, numbers are treated like words in Logo.

Another way to show the inequality of words and lists is demonstrated by the equal? and number? commands. Examples of these procedure commands and the screen displays are given as follows:

```
print equal? ‘a’ ‘A’
true
print equal? ‘one’ ‘1’
false
print equal? ‘turtle’ [turtle]
false
print number? 21
true
print number? [21]
false
print number? (21)
true
print number? 64 / 8
true
```

`Equal` reports true if *word/list1* and *word/list2* are the same, even if only one uses a capital letter. An examination of the `print number?` commands show that while a number can be included in a list, it is not equal to the number when used as a word, outside the list. Again, a group of characters used as a word is not equal to that group of characters inside a list.

Another command, identical (for example, `print identical? "a "A`), requires the characters of each input match exactly, even in terms of uppercase and lowercase characters. If these characters don't match, Logo outputs *false*.

While words and lists are not equal, they can be examined to determine if they are a member of each other. If *word/list1* is an element of *word/list2*, Logo reports *true*. If it is not an element, Logo reports *false*. The `print member?` *(word/list1)(word/list2)* instruction illustrates this relationship in the following examples and printed displays:

```
print member? ‘u’ ‘turtle’
true
print member? ‘turtles’ [Friendly turtles]
true
```

However, a character string is not an element of a list, even if a word within that list contains those characters:

```
print member? ‘u’ [Friendly turtles]
false
```

`Number` reports *true* if its input is a number. Above, when 21 is placed in brackets, it no longer is treated as a number. Brackets indicate a list which is a sequence of words. A list is not a

number and therefore is *false*. However, when parentheses are included around the number 21, the resulting display is *true*. Parentheses are not treated like square brackets (in other words, as indicating lists) and are used in Logo for number operations. `Print number?` shows the difference in Logo syntax between square brackets (lists) and parentheses (number operation).

To obtain a report on the number of elements in a word or list type `print count` *word/list* in the Command Center. If the input is a word, count reports the number of letters in the word:

```
print count 'Turtles'
7
```

If the input is a list, count reports the number of items (words or lists) in the list:

```
print count [Turtle Turtle Turtle]
3
```

To test the reorder program, type in the Command Center:

```
print Reorder [turtles and eggs]
```

The printed output in the textbox shows:

```
eggs and turtles
```

`Sentence` is the command or operation for putting lists together. `Sentence` is like the word operation for words. Sentence takes lists or words as inputs and puts them together into one list:

```
print sentence [Friendly turtles][move forward.]
Friendly turtles move forward.
pr sentence "Turtles [change color.]
Turtles change color.
pr sentence "Turtles "change
Turtles change
```

`Sentence` can only take two inputs (two words or two lists or a word and a list). If you type:

```
pr sentence "Turtles "change "color
```

…. you will get the following error message:

```
I don't know what to do with color.
```

You can use parentheses if you want to use more than two inputs with sentence, as in:

```
pr (sentence 'Turtles' 'change' 'color')
Turtles change color
```

Note that both the command `sentence` and all its inputs is included in the parentheses, but the pr command is not.

`Insert` *word*/list is a useful command when working with words and lists. Study the following example typed in the command center:

```
ct
insert 'friendly' insert [friendly turtle]
```

The `insert` command types the word friendly in the textbox and then, immediately after, types the list "friendly turtle" on the same line:

```
friendlyfriendly turtle
```

Another example followed by its textbox display is:

```
print 'friendly' insert [friendly turtle]
friendly
friendly turtle
```

The input to the next `print` or `insert` instruction will be typed immediately after the 'e' in turtle.

`Fput` or `lput` commands provide another way to insert a word or list at the beginning or end of a list. Study these examples:

```
print fput 'show' [the turtle]
show the turtle
print fput [show][the turtle]
[show] the turtle
print lput 'hides' [the turtle]
the turtle hides
print lput [hides] [the turtle]
the turtle [hides]
```

As shown above, `fput` (first put) places a *word/list* at the beginning of a list. `Lput` (last put) places a *word/list* at the end of a list.

Many primitives are available to the user when working with words and lists. These primitives are helpful for creating interactive lists and numbers programs.

Words and Lists in Logo Program Procedures

Use of word and list commands with print or insert in Logo programs is a way to manipulate and change character strings and words to create interesting graphic word displays. An understanding of the rules for writing words and lists will be helpful in writing program procedures that create visual word displays.

The following `TRIANGLE` program shows the use of words as inputs to procedures. Study the following recursive procedure:

```
to TRIANGLE :WORD
if :WORD = "[stop]
print :WORD
TRIANGLE butfirst :WORD
end
```

Create a textbox on the page. When `TRIANGLE "CYCLONE` is typed in the Command Center, the following word display appears in the textbox:

```
CYCLONE
YCLONE
CLONE
LONE
ONE
NE
E
```

The `TRIANGLE` program reduces the number of characters in a word by successively subtracting one. The procedure line `TRIANGLE` butfirst makes the word smaller by removing the first character. The if conditional statement stops the program when the word contains no characters (it is an empty word).

The following DOUBLE procedure shows the use of words as outputs to other commands or procedures. The DOUBLE procedure takes a word as an input and outputs the word twice, combined into one word:

```
to DOUBLE :WORD
output word :WORD :WORD
end
```

`Double` must report its output to a command that needs an input, for example, `print`. When `print DOUBLE "VISION` is typed in the Command Center, the following appears in the textbox:

```
VISIONVISION
```

Here's another example. Type:

```
print DOUBLE DOUBLE "VISION
VISIONVISIONVISIONVISION
```

With the TRIANGLE and DOUBLE procedures typed in the Procedures Pane of a project, typing the instruction, `TRIANGLE DOUBLE "CYCLONE` in the Command Center produces:

```
CYCLONECYCLONE
YCLONECYCLONE
CLONECYCLONE
LONECYCLONE
ONECYCLONE
NECYCLONE
ECYCLONE
CYCLONE
YCLONE
CLONE
LONE
ONE
NE
E
```

Various words may be typed into the TRIANGLE DOUBLE program to create neat graphic displays.

Here are some additional examples using words and lists in program procedures with recursion and variables, used to solve number factorials and change the order of words in a list. These adapted programs from the Logo Foundation (2012) are:

```
to Solvefactorial :num
if :num = 1 [output 1]
output :num * solvefactorial :num - 1
end
to Reorder :list
ifelse equal? count :list 1[output first :list]
           [output sentence reorder butfirst :list first :list]
end
```

A factorial is a nonnegative integer calculated as the product of all positive integers less than and equal to the number (e.g., 6 = 6x5x4x3x2x1 = 720). To test the factorial program, for example, type in the Command Center:

```
print Solvefactorial 6
```

The printed result in the textbox should be 720.

Interesting programs can be created using words and lists in recursive and variable procedures resulting in different list and number outputs. Access the *List of Lynx Primitive* user guide from the Help tool at the top of the Lynx website at https://lynxcoding.club/ to review the

List of Lynx Primitives along with *Lynx Vocabulary and Syntax* using words and lists from the Resource Materials.

Turtle Hint!
Remember to create a textbox by clicking on the "+" button for opening the window to select Text. You can now type in the Command Center `print` or `insert` to display words and lists in the textbox. For example, use the `repeat` command and `print` or `pr` commands, type in the Command Center:

```
repeat 20 [Pr [Hello Turtle!]]
```

The result shows the words "Hello Turtle" printed 20 times in a column going down the textbox. You can type `ct` (clear text) in the Command Center to erase the contents in the textbox.

Lynx Words and Lists Program Project

Turtle Activity 1
Practice with the `Print` statement and character strings and create different word and number outputs. For example, try:

1. `Print Sentence [Friendly turtles][move forward.]`
2. `Print (23 + 56 + 98) / 4`

View and save the results shown in the textbox.

Turtle Activity 2
Test the words and lists program procedures discussed in this section entering different words or numbers to show outcomes:

1. `to DOUBLE :WORD`
2. `to TRIANGLE :WORD`
3. `to Solvefactorial :num`
4. `to Reorder :list`

Turtle Activity 3
Create your own words and lists program procedures entering different words for manipulate words and lists and displaying outcomes.

Interactive Lists and Numbers Programs

Interactive programs are conversations between Logo and the computer user. Logo (and the Logo programmer) print a question in a textbox or the Command Center and wait for the question to be "read" or received by the computer user. The user then answers the question from the keyboard by typing a character (key) or line (word/list or number). This kind of interaction between the computer user and Logo is called interactive programming.

Working with Logo programs permits the programmer to pass information between procedures using numbers or words as either inputs or outputs. The ability to work with procedures as either inputs or outputs enables the Logo user to write interactive programs using lists. An interactive program is written as a conversation procedure. A procedure of this type means a question is printed and read or received on the page and an answer is typed with the keyboard.

Start by creating a textbox before trying these interactive words and lists programs.

```
to Meet
print [Hello and greetings ...]
print []
question [What is your name?]
make "name answer
print se [You have a very nice name,] answer
end
to Welcome
question [Hello, my name is Microchip. What is your
           favorite activity?]
announce [I like doing this too!]
print answer
end
```

Examples of the meet program written and executed in the Procedure Pane is displayed (Figures 25 and 26).

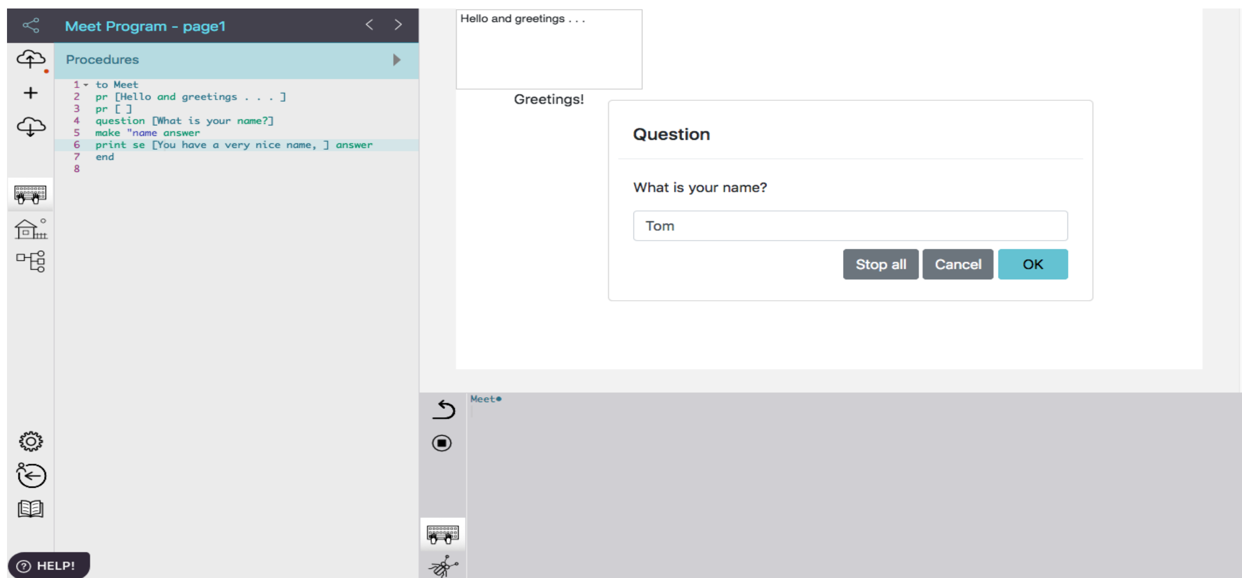


Figure 25. Demonstration of the Meet program presenting a question in the Lynx program.

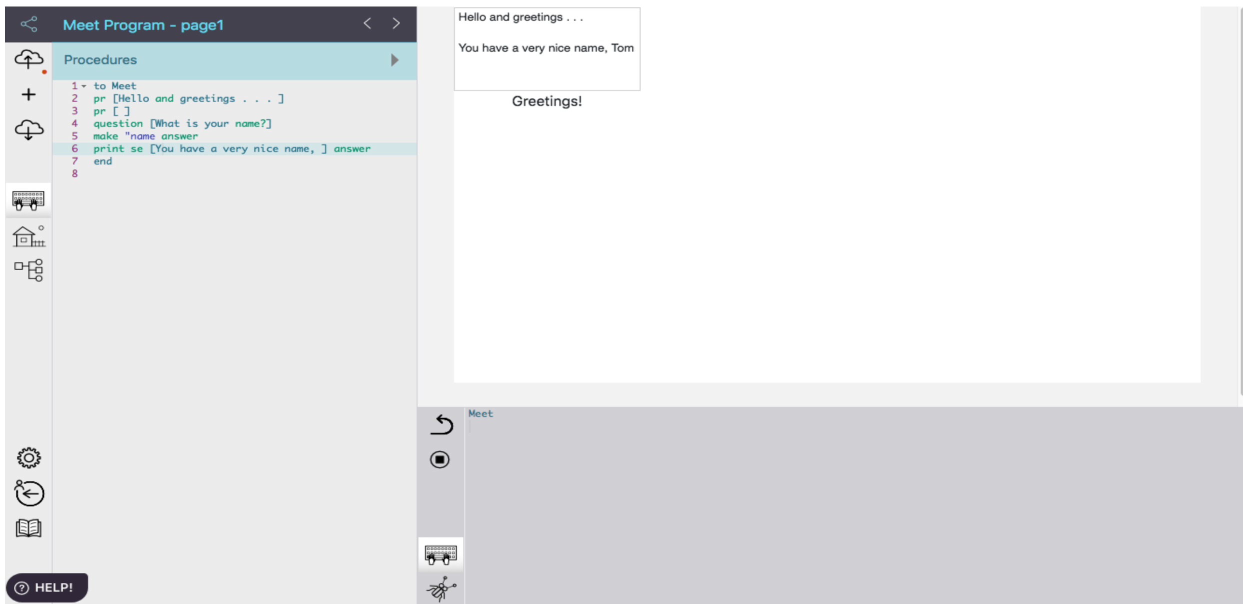


Figure 26. Demonstration of the Meet program displaying the final greeting in the Lynx program text box.

The command `question` opens a dialog box displaying the question and an area to type the answer. `Answer` reports what was typed in the dialog box. `Make` creates a variable, in this case called `name` and gives it a word or list, in this case, whatever `answer` reports, as its value. The quotation mark in front of the variable `name` in the `make` command lets Logo know you are creating a variable named `name`. `Print :name` means print the value of the variable called `name` in the textbox.

Use of words and list commands with `say` in Logo programs is way to add audio communication to a project. For example, for a talking program type the following procedure:

```
To TurtleTalk
Say [What do you want to talk about? I am a turtle
ready to speak!]
End
```

Experiment and change the name of the program and words to communicate the desired message for a Lynx project. An example of a student words and list project is Dudetalk (Figure 27).

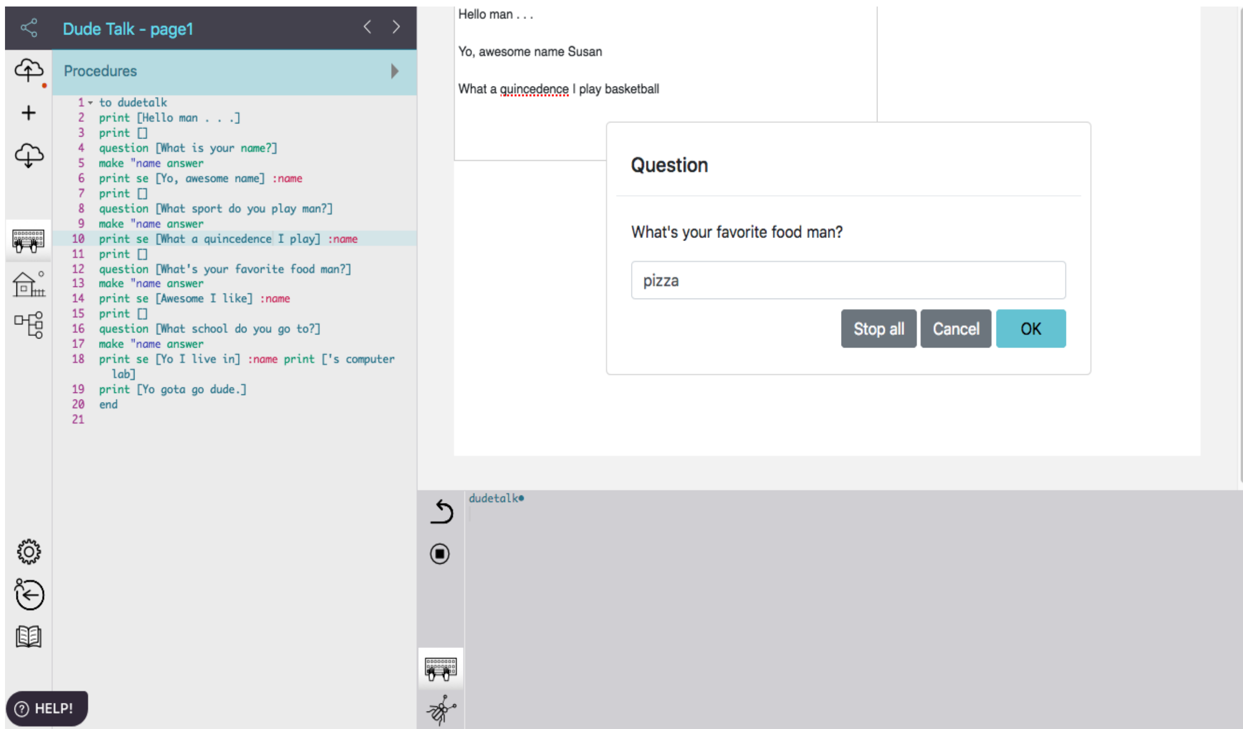


Figure 27. A student interactive words and lists project called Dudetalk. (Refer to appendix for program procedures.)

Lynx Interactive Lists and Numbers Program Project

Turtle Activity 1

Test the lists and numbers interactive procedures discussed in this section, entering different words and keystrokes for displaying outcomes.

1. To `Meet`
2. To `Welcome`
3. To TurtleTalk

Turtle Activity 2

Select from the Appendix the coding project Figure 27 (`dudetalk`) to copy and run in the Command Center. Debug and add program procedures to enhance the project.

Turtle Activity 3

Create your own lists and numbers interactive program procedures and test the program so it runs efficiently.

Applying Graphics, Animation and Interactive List Procedures for Developing Games

After gaining experience in the Lynx coding environment students will be ready to develop interactive games using the Logo language. Students with understanding to apply programming in creating graphics, animation, and using interactive words and lists procedures will have the tools to

develop a Logo based game. Some students may benefit from reviewing existing games created for project ideas or use of coding procedures to incorporate into their own interactive game.

Examples of interactive games can be found at the Lynx website at https://lynxcoding.club/. Sample Lynx programs to review can be found by scrolling to the tabs below the CREATE A LYNX PROJECT button. Suggested for the Learner Mode is:

- Math and Matches

For the advanced tab review the following projects:

- Throw the Dice
- Way Home

The Games tab has these program designs:

- Seeker Game
- Ha Ha Headlines

The Hackergal tab shows a number of project completed by girls during a Hackathon event, which may provide additional ideas to incorporate into an interactive game.

The **Help** tool at the top of the Lynx website has a pull down window to select **User Guides**. The downloadable pdf documents provide detail steps in how to create games and projects. You can select links provided under the Quick, Theme Based Activity Cards or Project Plans and Teacher's Notes titles (refer to Figure 28).

Quick, Theme Based Activity Cards
Geometric Fun
Interactive Greeting Card
How to Create Secret Codes
Birthday Match
Ecosystem
Create a Calculator App
Coin Toss Probability

Project Plans and Teachers' Notes
Story - Intro level for Students
Story - Teachers' Notes

Race simulation - Intermediate level for Students
Race simulation - Teachers' Notes

Video game maker - Advanced level for Students
Video game maker - Teachers' Notes

Figure 28. Help User Guide link of project ideas

Interactive Game Project

Turtle Activity 1

Create your own game by reviewing the Lynx web site of interactive projects.

Turtle Activity 2

Follow the directions in the User Guides to create a game project.

Appendix: Resources & Activities

Turtle Primitives

DIRECTIONS: Experiment with these commands in the Command Center and draw a shape or design a graphic. Commands do NOT need to be typed in capitals. If needed, write on the lines below to help you remember what the command does.

`fd` *number*	______________________________
`bk` *number*	______________________________
`rt` *number*	______________________________
`lt` *number*	______________________________
`cg/clean`	______________________________
`home`	______________________________
`ht/st`	______________________________
`pd`	______________________________
`pu`	______________________________
`pe`	______________________________
`setc` *number*	______________________________
`seth` *number*	______________________________
`show heading`	______________________________
`fill`	______________________________
`setpos[`*number1 number2*`]`	______________________________
`show pos`	______________________________
`setpensize` *number*	______________________________
`show pensize`	______________________________

Turtle Degrees

DIRECTIONS: Follow these activity steps and answer the questions to practice using degrees and making turtle right and left turns. Use the turtle clock, if needed.

I. Use your Turtle Clock and fill in the number of degrees the turtle should turn to point to the time shown.

1) 4:00 ________________ 2) 7:00 _______________

3) 10:00 _______________ 4) 1:00 ________________

5) 3:00 ________________ 6) 9:00 ________________

7) 6:00 ________________ 8) 12:00 _______________

9) 2:00 ________________ 10) 5:00 _______________

11) 8:00 _______________ 12) 11:00 ______________

II. Write down the hour the turtle is pointing after it makes these moves. The turtle returns to 0 degrees or the home position after making each turn.

1) rt 90 _______________ 2) rt 120 _______________

3) rt 60 _______________ 4) rt 300 _______________

5) rt 180 ______________ 6) lt 90 ________________

7) lt 180 ______________ 8) lt 150 _______________

9) lt 240 ______________ 10) lt 330 ______________

11) rt 30 ______________ 12) lt 60 _______________

13) lt 120 _____________ 14) rt 360 ______________

15) lt 360 _____________ 16) rt 150 ______________

17) rt 210 _____________ 18) lt 30 _______________

19) lt 210 _____________ 20) rt 330 ______________

* 21) rt 540 ____________ 22) lt 390 _____________

III. Write down the hour at which the turtle is pointing *after* it makes each *series* of moves. Return to the seth 0 degrees or home position after finishing the series of turns.

1) `rt 30 rt 30 rt 30` ________________
2) `lt 90 rt 90 lt 90` ________________
3) `rt 30 lt 90 rt 210` ________________
4) `lt 90 rt 180 rt 180` ________________
5) `rt 30 rt 120 lt 180` ________________
6) `rt 60 rt 60 rt 60` ________________
7) `rt 180 rt 180` ________________
8) `lt 360 rt 360` ________________
9) `rt 90 lt 180 lt 30 rt 210` ________________
10) `lt 60 lt 60 lt 60 lt 60 lt 60 lt 60` ________________
11) `lt 180 lt 90 rt 30 rt 60` ________________
12) `lt 270 rt 180 lt 60` ________________

IV. Questions

1. If the turtle is pointing to 4:00, how many more degrees does it have to move to reach 7:00?

____

2. You have turned the turtle rt 210. How many more degrees right should the turtle turn to reach 0?

____

3. How many rt 90 commands would get the turtle back to the same place?

____

4. Starting with the turtle at 12:00, give any three ways to get the turtle to reach 5:00. You can make more than one move.

____

5. A 90-degree turn is called a right angle. Ninety-degree angles are everywhere, such as the corner of a door. Name some examples of right angle objects you have seen.

____

Lynx Turtle Shapes

DIRECTIONS: For each shape fill in the table using the headings provided. Use the turtle clock, if needed. When finished answer the question at the bottom of the page.

Shape	Number of Sides	Number of Degrees for Each Turn	Repeat Statement
Triangle			
Quadrilateral			
Pentagon			
Hexagon			
Septagon			
Octagon			
Nonagon			
Decagon			
Circle			
*Other? ____________			

Question: What is the turtle rule or relationship between the number of turns (rt or lt) and repeat number?

____

____

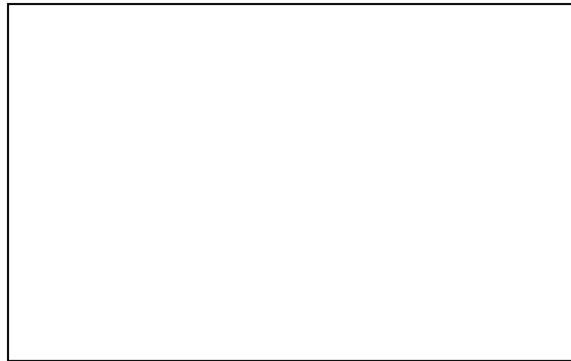
Repeat Predictions

DIRECTIONS: Complete the following activities by first writing the repeat procedure the each set of single commands. Next type the repeat instructions in the Command Center of a new page and draw the graphic outcome.

1. For each set of single commands write the **repeat** statement. Sketch the drawing of the graphic created in each space provided.

Single Commands — Drawing

```
cg
fd 100
rt 120
fd 100
rt 120
fd 100
rt 120
```



`repeat` ____

When finished drawing type cg.

Single Commands — Drawing

```
cg
fd 100
lt 90
fd 100
lt 90
fd 100
lt 90
fd 100
lt 90
```

`repeat` ____

When finished drawing type `cg`.

2. Draw a picture and predict what each of these repeat statements will tell the turtle to do. Type the repeat procedures in the Command Center then draw a picture of the

outcome. Before each repeat statement, the turtle starts in the home position or center of the page at the seth 0 position.

```
repeat 6 [lt 90 fd 10 rt 90 fd 10 bk 10 ht]
```

Predicted Drawing	Turtle Computer Drawing

Did your predicted and actual picture drawings match?

```
repeat 7 [rt 90 fd 10 lt 90 fd 15 bk 15 ht]
```

Predicted Drawing	Turtle Computer Drawing

Did your predicted and actual picture drawings match?

```
repeat 4 [fd 80 rt 90 fd 10 rt 90 fd 10 rt 90 fd 10 rt 90 bk 70 rt 90]
```

Predicted Drawing	Turtle Computer Drawing

Make changes to the above design by changing the size (`fd` *`number`*) and/or angles (example: `rt 90 to lt 90`).

3. Place different numbers in the repeat instruction below and type these in the Command Center. Change the `fd` to `bk` and/or `rt` to `lt` commands.

   `repeat` _____ [____]

What graphic design did you make?

A-Mazing

DIRECTIONS: Practice typing program procedures using the repeat command and your turtle orientation skills with this maze program. Type the maze program in the Procedures Pane and then run the program by typing, "maze" in the command center. Use the active turtle (e.g., green one) and type primitives (for example, `fd`, `bk` or `rt, lt`) in the Command Center to move between the maze squares in order to reach each of the four turtles. The `pd` command will show the path of the green turtle to all four black turtles. The command `setc 55` turns the turtle green. To view the maze program displayed in the Lynx windows refer to Figure 29).

```
to maze
pd repeat 4 [fd 50 rt 90] pu lt 90 fd 20
pd repeat 4 [fd 50 rt 90] pu lt 90 fd 20
pd repeat 4 [fd 50 rt 90] pu lt 90 fd 20
pd repeat 4 [fd 50 rt 90] pu fd 70
pd repeat 4 [fd 50 rt 90] pu lt 90 fd 20
pd repeat 4 [fd 50 rt 90] pu rt 90 fd 70
pd repeat 4 [fd 50 rt 90] pu lt 90 fd 20
pd repeat 4 [fd 50 rt 90] pu fd 70
pd repeat 4 [fd 50 rt 90] pu lt 90 fd 70 rt 90 bk 20
pd repeat 4 [fd 50 rt 90] pu lt 90 fd 20
pd repeat 4 [fd 50 rt 90] pu fd 70
pd repeat 4 [fd 50 rt 90] pu fd 70 lt 90
pd repeat 4 [fd 50 rt 90] pu fd 70
pd repeat 4 [fd 50 rt 90] pu fd 70
pd repeat 4 [fd 50 rt 90] pu rt 90 fd 70
pd repeat 4 [fd 50 rt 90] pu rt 90 fd 70 lt 90
pd repeat 4 [fd 50 rt 90] pu fd 70
pd repeat 4 [fd 50 rt 90] pu lt 90 fd 20
pd repeat 4 [fd 50 rt 90] pu fd 70 rt 180 pd stamp
pu setpos [-170 85] stamp
pu setpos [125 145] pd stamp
pu setpos [85 -95] pd stamp
pu home rt 90 pd setc 55
end
```

Turtle Questions to Ponder:

1. Were you able to reach the four turtles without bumping into the maze squares? Do you need more practice with understanding the length of turtle steps or turning using degrees?
2. What does the setpos primitive do? How is this command similar and different to the home command? What does the stamp command do?

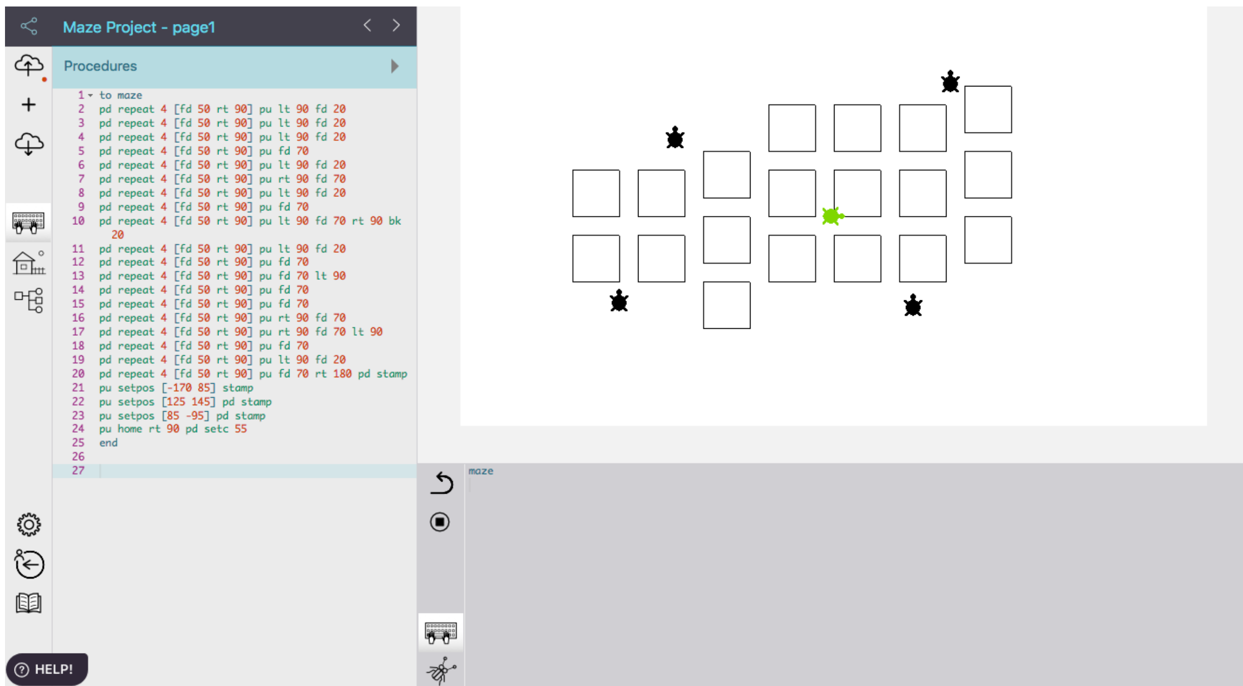


Figure 29. A green turtle pursuit to reach the four black turtles through a maze.

Additional Maze Activities

Turtle Activity 1

Change the setpos commands in the maze program to reposition the turtles in a different location. Try the maze again.

Turtle Activity 2

Maze Game: Use the maze program procedure to develop an interactive animated game.

Cognitive Monitoring Planning

DIRECTIONS: Follow the steps for planning a graphic and writing the Logo procedure into a working program.

Planned Graphic (Draw the picture you want on the computer.)

Shapes
(List the main shapes?)

Plan
(In sentences write a well planned turtle trip.)

Execute
(Write the turtle commands to make this graphic.)

to --

----

end

Outcome (Draw the picture the first time attempted.)

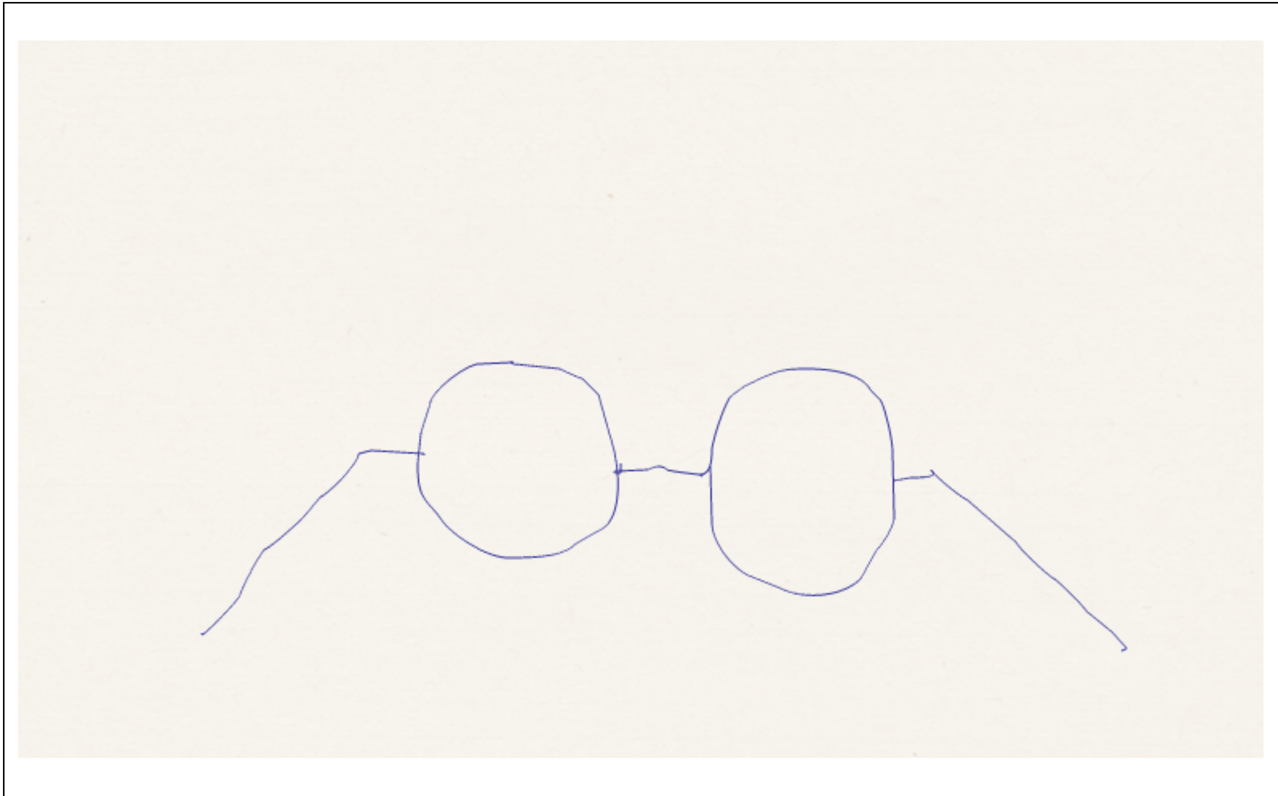
Does the planned graphic and outcome match? If yes, you are **finished**! If not, go on to follow the **Debugging Steps**.

Debugging Steps

1. Look at the difference between the picture and what the turtle draws.
2. Look for mistakes in your Logo program like misspelled commands, spaces between commands and numbers, or commands that are not correct.
3. Copy the program commands to the Command Center.
4. Press the **Return/Enter** key to run each command *line-by-line* to see how the graphic is being drawn. Erase and change command lines that are incorrect.
5. After the corrections are made, copy the procedure lines back to the Procedures Pane remembering to include the to *name* and end program lines.
6. Type the name of the program in the Command Center to see if the planned graphics and outcome graphics match. If the graphics do not match repeat debugging steps 1 – 6 again.

Cognitive Monitoring Student Project Example

Planned Graphic (Draw the picture you want on the computer.)

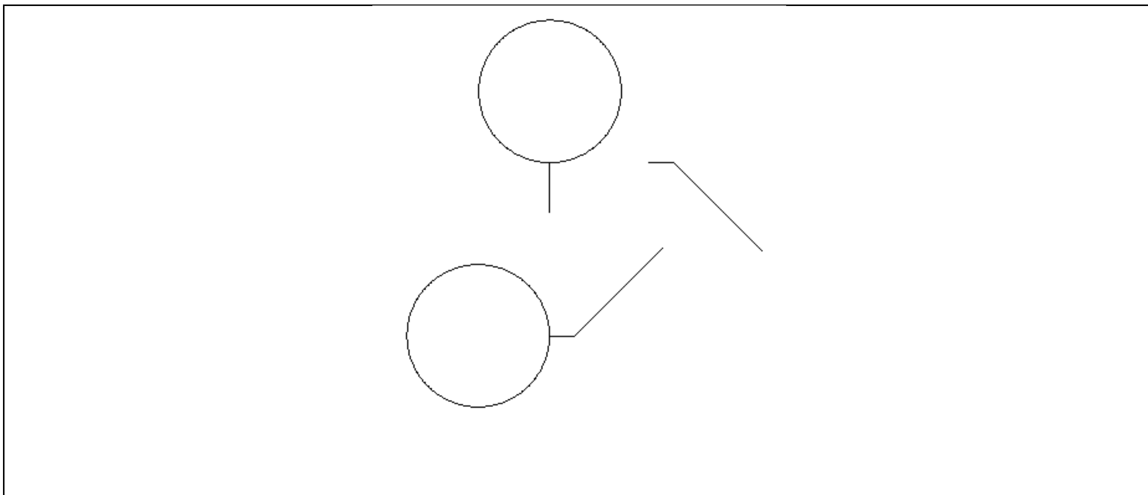


Shapes (List the main shapes?)	Lines and circles
Plan (In sentences write a well planned turtle trip.)	I will start at the right and make the frame to connect to the first lens. Another line will connect the next lens and then finish the frame.

Execute
(Write the turtle commands to make this graphic.)

```
to glasses2
rt 90
rt 45
fd 20
fd 100
bk 100
lt 225
pu
fd 100
pd
circle
lt 90
fd 40
pu
fd 100
pd
circle
lt 90
fd 20
lt 45
fd 100
end
```

Outcome (Draw the picture the first time attempted.)



Does the planned graphic and outcome match? If yes, you are **finished**! If not, go on to follow the **Debugging Steps**

Changing Procedures and Predicting Skills

DIRECTIONS: Before typing each Guess program make a predicted drawing and then draw the graphic created after the program name is typed in the command center.

```
to guess1
repeat 2 [fd 80 rt 90 fd 40 rt 90 fd 40 rt 90
fd 40 rt 90 bk 40 rt 180]
end
```

Guess1 Program

Predicted Drawing	Turtle Computer Drawing

```
to guess2
repeat 3 [fd 50 rt 120]
rt 120
repeat 3 [fd 50 rt 120]
rt 120
repeat 3 [fd 50 rt 120]
end
```

Guess2 Program

Predicted Drawing	Turtle Computer Drawing

```
to guess3
repeat 4 [fd 50 rt 90 fd 20 rt 90]
end
```

Guess3 Program

Predicted Drawing	Turtle Computer Drawing

```
to guess4
guess3 rt 90
guess3 rt 90
guess3 rt 90
guess3 rt 90
end
```

Guess4 Program

Predicted Drawing	Turtle Computer Drawing

Additional Optional Activity: Change the Turnit program into creating a new graphic idea or write your own guess program procedure.

```
to turnit
repeat 4 [fd 50 bk 50 rt 90]
end
```

Multiple Turtles

DIRECTIONS: Hatch four turtles. The first turtle hatched is turtle 1 or `t1` and the rest of the turtles are `t2`, `t3`, and `t4`. Experiment with these commands in the Command Center to learn their functions. Create a button and type the command stopall in the instruction line to stop the moving turtles on the screen.

`home`	________________
`clone "t1`	________________
`talkto [t1] pu fd 100 pd fd 100`	________________
`setc random 50`	________________
`clone "t2`	________________
`clone "t3`	________________
`talkto [t1 t3]`	________________
`pd repeat 4 [fd 50 rt 90] fd 100`	________________
`pd ask [t1 t3] [setc 25 rt 90]`	________________
`tto [t2 t4] pd bk 100`	________________
`repeat 3 [rt 120 fd 50]`	________________
`stamp rt 90 fd 100`	________________
`tto [t1 t4] setc 127 pd fd 150`	________________
`everyone [fd 50 lt 180 fd 50]`	________________
`everyone [glide 140 2]`	________________
`glide 50 1 glide 100 0.1`	________________
`show who`	________________

A Turtle Calculator Application

By typing the command `print` (`pr`) or `show`**,** followed by numbers and math symbols, the Logo can display solutions to math problems. Logo is able to perform calculator functions and the Logo primitives pr and show display the numbers and answers on the screen. Print displays the results in a textbox (created with the Create a Textbox tool in the Toolbar), while show displays the number or answer in the Command Center.

Here is a list of math symbols in Logo and what they do:

+	Adds
-	Subtracts
*	Multiplies
/	Divides
=	Reports *true* if its two inputs are equal, otherwise reports *false*
>	Reports *true* if the input on the left side of the symbol is greater than the input on the right, otherwise reports *false*
<	Reports *true* if the input on the left side of the symbol is less than the input on the right, otherwise reports *false*

Remember to put spaces between each symbol and its input numbers.

For example, if the following problems are typed into the Command Center, their answers will appear on the next line below the word show:

```
show 5369 + 89356
94725

show 90001 - 7483
82518

show 4390 * 8235
36151650

Show 141708 / 21
6748
```

The other math symbols are typed the same way, except that they will report *true* or *false* on the line below the word show**.** Examples of problems using the equal sign, greater than, or less than symbols are:

```
show 6 + 8 = 12
false

show 9 * 7 = 63
true

show 8 > 3
true

show 9 < 6 – 3
false
```

The `show not` instruction lets you find the opposite (inverse) condition of a problem used as input. For example, study the following problems and the displayed answers:

```
show not 1 = 2
true

show not 6 + 8 = 12
true

show not 8 > 3
false

show not "true
false
```

The `not` primitive reports inverse (opposite) of the problem input. These math symbols along with the show not instruction can be used in various ways to solve math problems.

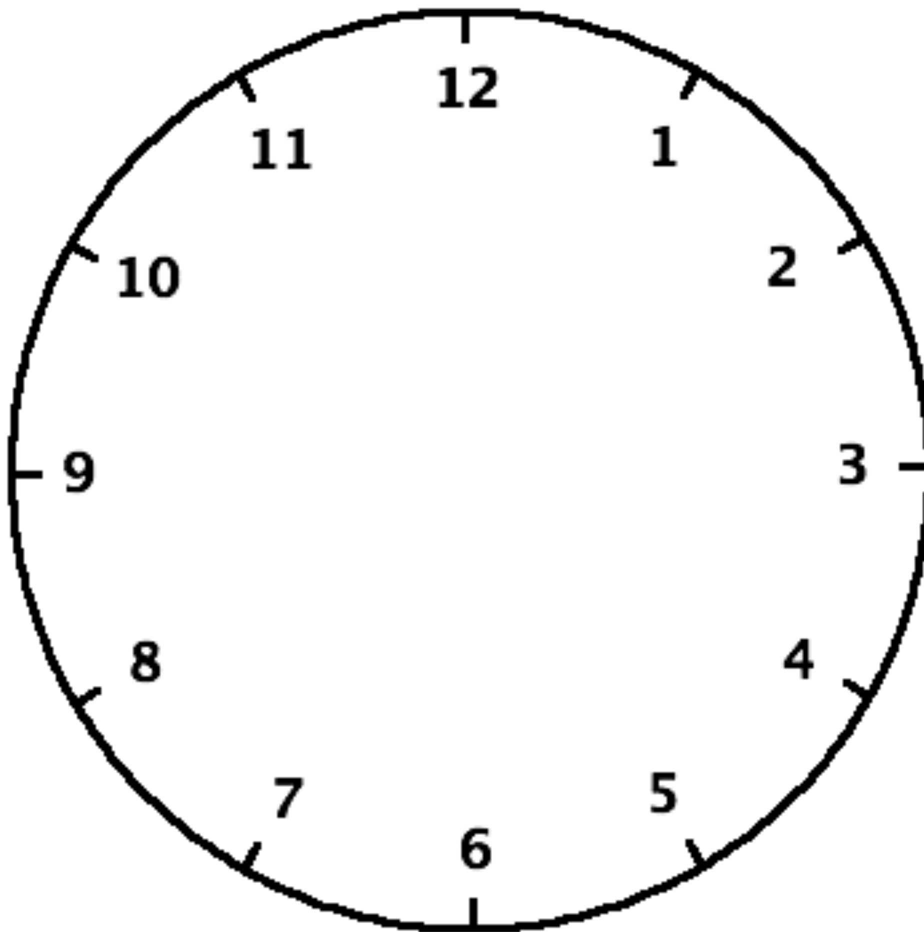
MicroWorlds Math Problems

Turtle Activity 1
Check your math assignment or apply the turtle words and lists commands to complete a computational project.

Turtle Activity 2
Create some math problems to solve, using print and show, and display the outcomes.

Turtle Degree Clock

DIRECTIONS: On the inside center circle of the clock write `rt` and outside the circle `lt`. Write on the turtle clock on the inside next to each hour line the `rt` (for right turns) number of degrees. Write on the outside of the clock circle next to each hour line the `lt` (for left turns) number of degrees. Begin above the number 12 (12:00) write 0 / 360 (for 0 degrees and 360 degrees). Questions: If a complete circle is 360 degrees then how many degrees turn is each hour going to change for rt turns and lt turns?



Turtle Hint!
There are 30 degrees between each hour of a clock. Start with 0 degrees at 12:00 and make a complete turn returning to 12:00 resulting in 360 degrees. If the turtle starts at 0 degrees and turns right 90 degrees or rt 90 would it be facing 3:00? If the turtle turns the opposite direction returning to 0 degrees and turns left 90 degrees or lt 90 would it be facing 9:00?

Logo Program Procedures Learning Models from Figures

Project Ideas to Edit and Debug
DIRECTIONS: Select a coding project to run in the Lynx program. Follow these steps:
1. Copy and paste the program in the Procedures Tab window and test run the program in the command center.
2. Debug the program by pasting procedures in the command center and running the program line by line to find the error.
3. Correct program line errors by recopying the program or editing in the Procedures Tab.
4. Test run the program again to check for successful outcomes, if not repeat the debugging steps.
5. Edit by changing and enhancing the program procedures to create a new project or adaptation.

Figure 1. Information display of Lynx layout features available from the Ecosystem activity and also from other cards on the Lynx web site.

Figure 2. Information window display example for `repeat` primitive when hovering over command.

Figure 3. Refer to the gear program procedures in turtle activity project.

Figure 4. Student **swim goggle** created with turtle primitives and repeat instructions (written as program procedures).

```
To SwimGoggles
setc 85
setpensize 15
Repeat 540 [Fd 1 Rt 1]
Rt 270 Fd 110
Rt 90 Fd 10
Rt 90 Fd 110
Rt 90 Fd 10
Rt 90 Fd 110
Rt 270 Repeat 540 [Fd 1 Rt 1]
Rt 270 Repeat 180 [Fd 1 Rt 1]
Fd 325
Repeat 170 [Fd 1 Rt 1]
end
```

Figure 5. Student **smileyface** program created with turtle primitives and repeat instructions.

```
to smileyface
pd setsize 9 setpensize 2
repeat 360 [fd 1 rt 1]
repeat 360 [fd 1 lt 1]
pu bk 60 pd
repeat 360 [ fd 0.2 rt 1]
pu bk 75
pu rt 90 fd 90 lt 90 fd 10
pd repeat 180 [bk 1 rt 1]
lt 180
pu fd 80 fd 93 pu bk 45lt 90 fd 45
pu rt 180 fd 90 lt 90 fd 43
pd bk 90
pu fd 45 lt 90 fd 10
pd fill
pu setpos [-92 -43]
pd rt 90 fd 90
seth 200
pu fd 50 fill pu fd 80
seth 340
pd repeat 360 [ fd 3.5 rt 1]
pd setc 45 rt 90 fd 10 fill
pu setc 15 setpos [11 -57] fill
end
```

Figure 6. Student **arrow** program using turtle commands with repeat instructions.

```
to arrow
pd setc 105
lt 90
repeat 3 [rt 120 fd 100]
bk 25
seth 0
bk 100
repeat 2 [fd 100 rt 90 fd 50 rt 90]
pu
rt 45
fd 10
fill
seth 0
fd 100
fill
ht
end
```

Figure 7. Lynx student project showing a **computer laptop** program.

```
to laptop
rectangle
rt 110
keyboard
pu rt 20 fd 100 rt 70 fd 5 lt 90 fd 5
screen
pu bk 2 setc 14 fill
pu fd 2
picture
end

to rectangle
setc 9 pd
repeat 2 [fd 100 lt 90 fd 160 lt 90]
end

to keyboard
fd 100 rt 160 fd 160 rt 20 fd 100
pu lt 10 bk 30
setc 84 fill
pu fd 30 rt 10
pd setc 9
repeat 5 [bk 20 rt 160 fd 160 bk 160 lt 160]
lt 20
repeat 8 [bk 20 rt 20 fd 100 bk 100 lt 20]
end

to screen
pd setc 9
repeat 2 [fd 150 rt 90 fd 90 rt 90]
end

to picture
rt 45 fd 40 lt 45 fd 10 rt 90 fd 10
pd setc 9
repeat 360 [fd .5 lt 1]
ht
end
```

Figure 8. A student modular **clock** program with circle and number subprocedures.

```
to clock
pd
circle
repeat 180 [fd 2 rt 1]
three
rt 90
repeat 180 [fd 2 rt 1]
nine
repeat 90 [fd 2 rt 1]
twelve
pu
home
repeat 270 [fd 2 rt 1]
pd
six
pu
home
ht
end

to circle
Repeat 360 [fd 2 rt 1]
end
```

```
to three
rt 90 fd 10 pd rt 90 fd 10 bk 20
fd 10 lt 90
pu fd 10
pd rt 90 fd 10 bk 20 fd 10 lt 90
pu fd 10
pd rt 90 fd 10 bk 20 fd 10
pu rt 90 fd 30
pd
end

to nine
rt 90 fd 10 lt 90 fd 10 bk 20 fd
10
pu rt 90 fd 10
pd lt 60 fd 12 bk 24 fd 12 lt 60
fd 12 bk 24 fd 12 lt 60
pu fd 20 rt 90
pd
end

to twelve
rt 90 fd 10 rt 90
pu fd 20 lt 120
pd fd 24 bk 12 rt 60 bk 12  fd 24
bk 24 lt 120
pu fd 6
pd rt 90 fd 20 lt 90
pu fd 10 lt 90
pd fd 20
end

to six
rt 90 fd 10 lt 90
pu fd 10
pd rt 67 fd 24 bk 24 rt 46 fd 24
rt 67
pu fd 5 rt 90
pd fd 21
end
```

Figure 9. A student modular program with **smiley, snake, and peace** subprocedures

```
to everything
smiley
st pu fd 150 rt 90 fd 50 rt 90
snake
pu fd 200 rt 90 fd 200
peace
end

to snake
st pd setc 9
repeat 3 [rt 45 fd 20]
lt 135 repeat 3 [rt 45 fd 20]
lt 135 repeat 3 [rt 45 fd 20]
rt 45 fd 15 rt 135
repeat 3 [fd 20 lt 45]
rt 135 repeat 3 [fd 20 lt 45]
rt 135 repeat 3 [fd 20 lt 45]
rt 135 fd 20 lt 45 fd 20 rt 45
repeat 4 [rt 45 fd 10]
fd 14 bk 24 lt 225 fd 10 lt 45 fd
10 lt 45 fd 5
rt 90 fd 10 rt 45 fd 5 bk 5 lt 90
fd 5 bk 5
rt 45 bk 10
pu bk 5
pd setc 65 fill
ht
end
```

```
to peace
st pd setc 9
repeat 9 [rt 45 fd 50]
rt 45 fd 25 rt 90 fd 120 bk 60 rt
45 fd 60 bk 60 lt 90 fd 60
pu lt 45 bk 10 setc 24 fill
pu lt 90 fd 50 setc 44 fill
pu lt 90 fd 75 setc 85 fill
pu lt 135 fd 50
setc 113 fill
ht
end

to smiley
st pd setc 9
repeat 360 [fd 2 rt 1]
rt 90
pu fd 50
pd fd 115 rt 90
repeat 180 [fd 1 rt 1]
pu fd 40 pd
repeat 360 [fd .3 rt 1]
repeat 360 [fd .15 rt 1]
pu rt 90 fd 80 lt 90 pd
repeat 360 [fd .3 rt 1]
repeat 360 [fd .15 rt 1]
pu rt 90 fd 5 setc 9 fill
pu fd 20 setc 95 fill
pu bk 100 setc 9 fill
pu fd 20 setc 95 fill
pu fd 20 setc 44 fill
pu rt 90 fd 50
setc 15 fill
ht
end
```

Figure 10. A student modular **neighborhood** program with house, tree, window, circle, and square subprocedures.

to neighborhood pd setc 9 house pu seth 0 setpos [-164 -180] setc 9 house end to circle pd repeat 39 [fd 10 rt 10] pu rt 50 fd 30 fill end to square pd repeat 4 [fd 150 rt 90] end	to house pd square setc 65 rt 90 fd 300 lt 90 fd 75 lt 90 fd 150 lt 90 fd 75 lt 90 fd 20 lt 90 fd 50 rt 90 fd 115 rt 90 fd 50 rt 90 fd 150 fd 50 rt 90 fd 50 lt 90 fd 30 lt 90 fd 50 rt 90 pu fd 20 rt 90 fd 100 lt 90 fd 10 rt 90 window rt 90 pu fd 75 lt 90 window pu fd 40 fd 10 lt 90 fd 100 rt 140 pd fd 80 rt 70 fd 105 pu rt 100 fd 150 rt 50 fd 100 lt 90 fd 30 rt 180 pd tree end to window pd repeat 4[fd 30 rt 90] end to tree pd fd 40 lt 90 fd 20 lt 90 fd 40 lt 90 fd 20 lt 90 fd 40 lt 90 fd 10 circle end

Figure 11. A student modular broken key **piano** program with various position piano part subprocedures.

```
To PIANO
PosPiano
BodyPiano
CovPiano
WhiteKeys
BlackKeys
PadPiano
GreenPia
End

to clear
cg setc 9
end

To PosPiano
st setc 9 cg pu lt 90 fd 250
End

To BodyPiano
Rt 90 pd fd 60 rt 90 fd 500 rt 90
fd 60
pu bk 60 rt 180 pd
End

To CovPiano
setpensize 2
Repeat 30 [ fd 1 lt 1 fd 1 lt 2 ]
Repeat 30 [ fd 2 rt 1 ]
Fd 30 Repeat 10 [ fd 1 lt 0.5 ]
Fd 100
Repeat 150 [fd 2 lt 0.5]
Repeat 25 [ fd 1 lt 1.5]
Fd 10 Lt 8 Fd 25
End
```

```
To BlackKeys
Repeat 19 [fd 33 rt 90 fd 10 rt 90
fd 33 lt 90
Fd 15 lt 90]
Fd 33 rt 90 fd 10 rt 90 fd 33 lt
90
Lt 90
Fd 20
Pu
Lt 90 fd 5
Repeat 3 [fill fd 22 fill fd 5
fill]
Bk 8 Fill fd 5 Fill Fd 22 fill fd
22 fill
fd 5 fill fd 22 fill fd 24 fill
fd 5 fill fd 22 Fill fd 24 fill
fd 5 fill fd 22 fill fd 24 Fill
fd 5 fill fd 22 fill fd 21 fill
fd 5 fill Fd 22 fill fd 24 fill
fd 5 fill fd 22 fill fd 24 fill
fd 5 fill fd 22 fill fd 22 fill
fd 5 Fill
Pu fd 16.5 rt 90 ht fd 40
End

To PadPiano
setpensize 1
St rt 90 pd fd 500 ht rt 90 fd 3
rt 90 Pu fd 5 fill st
End

To WhiteKeys
setpensize 1
seth 180
pu fd 60 bk 5 lt 90
pd
Repeat 50 [fd 10 lt 90 fd 55 bk 55
rt 90]
lt 90  fd 55
pu lt 90 fd 15 lt 90
pd
End

To GreenPia
pu Lt 45 fd 100 setc 66 fill
End
```

Figure 12. A student modular recursive named **TX2** superprocedure calling **g4** subprocedure.

<table>
<tr>
<td>

```
to TX2
g4
fd 50
g4
TX2
END

to guess3
repeat 4 [fd 50 rt 90 fd 20 rt 90]
end
```

</td>
<td>

```
to g4
guess3 rt 90
guess3 rt 90
guess3 rt 90
guess3 rt 90
end
```

</td>
</tr>
</table>

Figure 13. A student modular recursive superprocedure, including **random color** changing command, calling shape subprocedures.

<table>
<tr>
<td>

```
to shapes
setc random 200
circle
triangle
square
bk 10
rt 45 shapes
end
```

</td>
<td>

```
to triangle
repeat 3 [fd 100 rt 120]
end

to circle
repeat 360 [fd .5 rt 1]
end

to square
repeat 4 [fd 150 rt 90]
end
```

</td>
</tr>
</table>

Figure 14. A student modular variable program procedures creating different **shoe** colors.

to shoe :shoec :skin :backg setpensize 2 setc :shoec rt 90 base 150 30 toe 1 seth 0 repeat 90 [fd .5 lt 1] seth 180 fd .5 * 60 seth 270 base 70 .1 * 03 seth 180 base 50 .0001 * 0.3 seth 150 fd 20 seth 0 pu fd 30 fill setc :skin pu fd 50 bk 20 fd 10 pd fd 100 rt 90 fd 30 fd 20 rt 90 fd 100 rt 90 fd 50 seth 120 seth 60 fd 20 fill setc :backg seth 270 pu fd 100 pd fill ht end	to base :length :width repeat 3 [fd :length lt 90 fd :width lt 90] end to toe :size seth 0 repeat 90 [fd :size lt 1] end

Figure 15. A student **solar eclipse** variable program with subprocedures.

to solareclipse pd setc 7 fill home setc 9 blackcircle lt 90 fd 100 rt 90 pd circle 2 ht pu fd 200 end	to circle :size Repeat 360 [fd :size rt 1] end to blackcircle circle 2 pu rt 90 fd 100 fill home end

Figure 16. A student modular **outfit** procedures with assigned variable shirt values.

```
to outfit
shirt 100
pants
ht
end

to shirt  :Size
pd Fd :Size Rt 90 Fd :Size Rt 90
Fd :Size
rt 90 Fd :Size Rt 90 Fd 100 lt 90
pu fd 10 lt 45
pd seth 180 repeat 4 [fd 30 rt 90]
seth 90 fd 120 repeat 4 [fd 30 rt
90]
bk 100 repeat 180 [fd .25 rt .5]
lt 160 fd 30
pu rt 180 fd 25 fd 20
setc 93
end
```

```
to pants
fd 10 rt 90 fd 5 lt 90 fd 30 lt 25
fd 21
pd fd 99 lt 90 fd 20 lt 90 fd 100
pu fd 5
pu lt 90 fd 20 lt 90 fd 20 lt 90
fd 20
fd 20 lt 90 fd 20
pd bk 100 rt 90 fd 17 lt 90 fd 100
pu fd 10
fill
end
```

Figure 17. A student modular **crayons** procedures with assigned variable values.

```
to crayons
pd
crayon 119 116
setc 9
lt 90 fd 20 rt 90 fd 40 lt 90
pd crayon 69 64
ht
end
```

```
to crayon :wax :wrapper
setpensize 2
fd 200 rt 45 fd 30 rt 90 fd 30 rt
45
fd 200 rt 90 fd 42 rt 90 fd 25 rt
90
fd 42 lt 90 fd 150 lt 90 fd 42 bk
30 rt 90
pu  fd 20
pd setc :wax fill
pu bk 40 lt 180 setc 9
pd fd 100 repeat 180 [fd .2 rt 1]
fd 100 repeat 180 [fd .2 rt 1]
pu rt 45 fd 10
pd setc :wax fill lt 45
pu fd 130 fill
bk 20 setc :wrapper
fill
end
```

Figure 18. A student modular variable desk program using assigned variable values.

```
to desk
pu lt 90 fd 150 lt 90 fd 50 rt 90
pd setc 9
repeat 2 [rt 90 fd 200 rt 90 fd
300]
leg 70
pu fd 2 rt 90 bk 2 setc 23 fill fd
2 lt 90 bk 2 fd 10 lt 90 fd 2 setc
23 fill bk 2 rt 90 bk 10
pd setc 9 bk 265
leg 70
pu fd 2 rt 90 bk 2 setc 23 fill fd
2 lt 90 bk 2
fd 10 rt 90 bk 2 setc 23 fill fd 2
lt 90 bk 10
pd setc 9
rt 90 fd 158 lt 90
leg2
pu lt 90 fd 10 lt 90 fd 2 setc 23
fill bk 2 rt 90 bk 10 rt 90
pu lt 90 fd 30 rt 90 bk 2 setc 23
fill fd 2 lt 90 bk 30 rt 90
pu
fd 30 lt 90 fd 30
pd setc 9
pencil
ht
end
```

```
to leg :side
lt 110 fd :side lt :side fd 10 lt
110 fd :side bk :side rt 20 fd 20
lt 20 fd 48 lt :side
end

to leg2
lt 110 fd 50 lt 70 fd 10 lt 110 fd
80 bk 80 rt 20 fd 20 lt 20 fd 75
lt 70
end

to pencil
fd 50 rt 20 fd 10 rt 140 fd 10 rt
20 fd 50 rt 90 fd 7 rt 90 fd 10 rt
90 fd 7 pu bk 2 lt 90 bk 5
setc 133 fill
pu fd 5 rt 90 fd 2 lt 90 fd 40
pu bk 5 rt 90 bk 10
pu fd 7
setc 45 fill
pu bk 2
setc 9 lt 90 fd 4 rt 90 pd fd 5
pu bk 4 lt 90 fd 3
setc 23 fill
pu fd 30
setc 27 fill
end
```

Figure 19. A student variable project clearing and creating medieval colored **swords**.

```
to swords :c
setc :c
fill
repeat 4  [sword 125 50]
end

to circle
setc 104
repeat 43 [fd 2 rt 10]
end
```

```
to sword :s :h
circle
lt 70 setc 14 fd 30 lt 87
fd :h rt 170
fd :h lt 83
setc 139
fd :s
rt 20 fd 14 rt 139  setc 139 fd 14
rt 20
setc 25 fd :s
lt 88 setc 45
fd :h rt 170
fd :h lt 81
setc 85 fd 30 lt 90 pu fd 30 pd
end
```

Figure 20. Refer to the Rundog program in the Animating Turtle Shapes section of the text.

Figure 21. Refer to the text program procedures of a Lynx race animation using a control buttom to adjust the speed of one Lynx.

Figures 22. & 23. Refer to the directions and window figure displays for adding a button and slider to the Lynxrace animation.

Figure 24. Refer to the program procedures in the text of two Lynx racing at various speeds with the addition of a background shape.

Figures 25. & 26. Refer to the Meet program in the Interactive and Numbers Programs section of the text.

Figure 27. A student interactive words and lists project called **Dudetalk**.

```
to dudetalk
print [Hello man . . .]
print []
question [What is your name?]
make "name answer
print se [Yo, awesome name] :name
print []
question [What sport do you play man?]
make "name answer
print se [What a quincedence I play] :name
print []
question [What's your favorite food man?]
make "name answer
print se [Awesome I like] :name
print []
question [What school do you go to?]
make "name answer
print se [Yo I live in] :name print ['s computer lab]
print [Yo gota go dude.]
end
```

Figure 28. Help User Guide link of project ideas refer to the Lynx website at https://lynxcoding.club/ for examples of interactive games.

Figure 29. Refer to the program procedures provided in the A-Mazing activity.

Guide References and Resources

Abelson, Harold. (1982). *Apple Logo*. New Hampshire: BYTE/McGraw-Hill.

Billstein, Rick. (1982). "Learning Logo and liking it." *The Computing Teacher*, 10(3): 18-20.

Corbosiero, Louis J. (1986). *The teaching of grade 7 geometric concepts using Logo*. Needham, Mass: Needham Public Schools. (ERIC Document Reproduction Service No. 286 715)

Cuneo, Diane O. (1985). *Young children and turtle graphics programming: Understanding turtle commands*. Paper presented at the Biennial Meeting of the Society for Research in Child Development, Toronto, Ontario, Canada. (ERIC Document Reproduction Service No. ED 260 800)

Delclos, Victor R. (1984). *Teaching thinking through Logo: The importance of method*, (Report No. 84.1.2.) Nashville, Tennessee: Learning Technology Center. (ERIC Document Reproduction Service No. ED 262 756)

Friesen, Chuck and others (1984). *One key Logo and hands-on activity cards*. Lincoln, Nebraska: Nebraska State Dept. of Eduction. (ERIC Document Reproduction Service No. ED 265 833)

Logo Computer Systems Inc. (2020). *Create a Working Ecosystem with Lynx*. [Lynx Computer software user guide]. Retrieved from: https://lynxcoding.club/

Logo Computer Systems Inc. (2020). *Getting Started with Lynx Basic Techniques to Get You Started*. Volume 19 [Lynx Computer software user guide]. Retrieved from: https://lynxcoding.club/

Logo Computer Systems Inc. (2020). *List of Lynx primitives*. Volume 1.3 [Lynx Computer software user guide]. Retrieved from: https://lynxcoding.club/

Logo Computer Systems Inc. (2020). *Lynx Vocabulary and Syntax*. Volume 1 [Lynx Computer software user guide]. Retrieved from: https://lynxcoding.club/

Horn, M. and Boe, T. (1985). *Apple Logo in the classroom*. Minnesota: Minnesota Educational Computing Consortium.

Hunter, Beverly. (1983). *My students use computers: Learning activities for computer literacy*. Reston, Virginia: Reston Publishing Company. (ERIC Document Reproduction Service No. ED 237 060)

Kurland, Midian and Pea, Roy. (1984). *Logo programming and thedevelopment of planning skills*. (Report No. 16). New York, N.Y: Bank Street College of Education.

Lee, MiOk. (1991). *Effects of guided Logo programming instruction on the development of cognitive monitoring strategies among college students*. Unpublished Ph.D. dissertation, Iowa State University, Ames, Iowa.

Lee, P. and Mitchell, M. (1985). "Demystifying Logo recursion: a storage process model of embeded recursion." *The Computing Teacher*, 12(5): 197-208.

Logo Computer Systems, Inc. (1986). *Logo Writer reference guide*. New York, N.Y: Logo Computer Systems, Inc.

Logo Computer Systems, Inc. (1986). *Learning with LogoWriter*. New York, N.Y. Logo Computer Systems, Inc.

Logo Computer Systems, Inc. (1986). *LogoWriter teacher's manual*. New York, N.Y. Logo Computer Systems, Inc.

Logo Foundation. (2012). *The Logo Programming Language*. Accessed on May 27, 2012 @ http://el.media.mit.edu/logo-foundation/logo/programming.html

Louie, Steven. (1985). *Locus of control among computer-using school children*. A report of a pilot study. Tucson, Arizona: Natinal Advisory Council for Computer Implementation in Schools. (ERIC Document Reproduction Service No. ED 260 692)

Logo Computer Systems Inc. (2020). Lynx [Computer software]. Retrieved from: https://lynxcoding.club/

Martin, Kathleen and Riordon, Tim. (1984). "Polyspirals." *The Computing Teacher* 11(6): 53-55.

Martin, Max. (1985). "Recursion -- a powerful, but often difficult idea." *Computers in the Schools*, 2 (2/3): 209-217.

Nolan, Pat and Ryba, Ken. (1986) *Assessing learning with Logo*. Eugene, Oregon: International Council for Computers in Education, University of Oregon. (ERIC Document Reproduction Service No. ED 290 461).

Papert, Seymour (1980). *Mindstorms*. New York, N.Y. Basic Books Inc., Publishers.

Stager, Gary (2007). *Planet Papert articles by and about Seymour Papert*. Accessed on July 12, 2007 @ http://www.stager.org/planetpapert.html

Temple, Michael and others. (1985). *The ECCO Logo project: materials for classroom teachers and teacher trainers*. Eugene, Oregon: International Council for Computers in Education, University of Oregon. (ERIC Document Reproduction Service No. ED 288 487).

Thomas, Elearnor M., and Thomas, Rex A. (1984). Exploring geometry with Logo. *Arithmetic Teacher*, 32(1): 16-18.

Torgerson, Shirley and others. (1984). *Logo in the Classroom*. Eugene, Oregon: International Council for Computers in Education, University of Oregon. (ERIC Document Reproduction Service No. ED 248 847).

Walsh, Thomas E. (1993). The implementation and evaluation of a sequential, Structured approach for teaching LogoWriter to classroom teachers. *Journal of Educational Technology Systems* 21(4): 343-362.

Walsh, Thomas E. (1994). "Facilitating Logo's potential using teacher-mediated delivery of instruction: A literature review." *Journal of Research on Computing in Education*, 26(3): 322-335.

Watt, Dan. (1983). *Learning with Logo*. New York, N.Y. McGraw Hill.

Webb, J., Martins, P. Holmes, M. (1984). *Explorers Guide to Apple Logo*. Hasbrouck Heights, New Jersey: Hayden Book Co.

Yoder, S. (1988). *Introduction to Programming in Logo using LogoWriter*. Eugene, Oregon: International Council for Computers in Education.