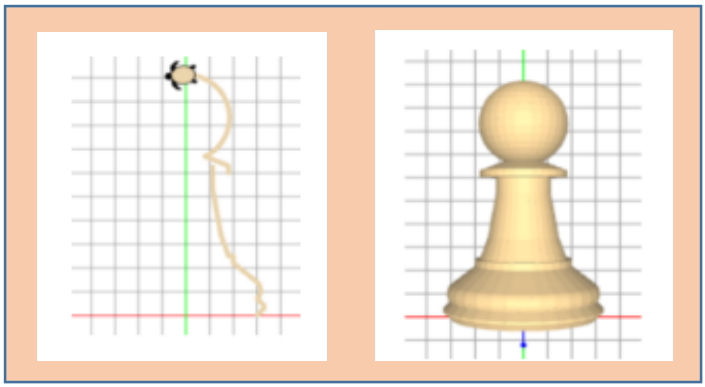
nclab
More Than Coding

TINA 2 PROGRAMMING COURSE

LESSON PLANS

REVISION: MARCH 28, 2016



OVERVIEW

Tina 2 is a series of lessons designed to teach students computational thinking and computer programming as applied to graphics. The language itself is a simplified version of Python, which is used extensively in engineering, science and design work. The basic concepts are common to all programming languages.

Tina 2 is a continuation of Tina 1, so students should already be able to write programs that draw complex designs using loops, nested loops, and variables. In Tina 2, students will learn how to write parametric variables and functions, and create arcs, solids, surfaces and shells. In Tina 1, students created 3-D objects by extruding two-dimensional designs: now they will learn how to create solids, surfaces and shells by rotating about the y-axis. The ability to manipulate variables and functions is powerful: students bring to life concepts that they learn in mathematics and apply them to design.

Tina 2 can be learned independently or under the guidance of a teacher. The course is divided into four sections, with eight levels in each section. Each level builds on the previous one and includes a tutorial and hints if students are stuck. There are links to YouTube videos that explain the steps. The online textbook, available from a drop down menu, describes the program and technical information in detail.

Although the program stands on its own, the value of the lessons is greatly enhanced by classroom discussion and solution sharing. At any stage of the course, students can freely create and test designs. By experimenting with the shapes, they will develop visual-spatial abilities and gain a deeper understanding of algebra and geometry.

Students can save designs to their own NCLab folder, publish and share links to the designs, or submit them to NCLab for display on the Gallery page. Best of all, students can create an STF file to print out their 3D designs!

EQUIPMENT AND ACCOUNTS REQUIRED:

- **Personal computers or tablets, with Internet access:** one per student. Both PC and MAC platforms are supported. Tablets can also be used. Preferred browsers are Google Chrome or Firefox.
- **Projector or Smartboard** (optional but recommended) attached to a computer for demonstration or modeling
- **Accounts:** The Tina Course requires individual accounts for each student. Visit the FAQ page for information on free and paid accounts. https://nclab.com/faq/ Have names and passwords ready on Day 1 to make logging on a smooth process (small cards with this information can be passed out to each student)
- **Teacher Accounts:** It is helpful to have two teacher accounts: one for learning the program and maintaining progress as a teacher; and the other for demonstrating the program. This second account will look like what the students encounter and can be cleared as needed for each class that is taught.
- **Progress monitoring:** Students accounts associated with a teacher can be progress monitored from the teacher's NCLab desktop.
- **The teacher textbook** can be downloaded as a .pdf file from the Resources page https://nclab.com/resources/
- **The solution manual**, which includes solutions for both Tina 1 and 2, can be accessed by following this link: https://docs.google.com/document/d/1iPAfL11RzLI353YDsDyeYGc-U0wR_iB81RrpJrdpn5M/edit
- **YouTube videos:** some schools block YouTube, so the demonstration videos may need to be unblocked by an administrator to make them available to students
- **Publishing:** Students should have a way to share a link to their games with others, such as a shared folder on a network drive; class or student wikis, web pages, blogs or email accounts; commercial networks such as Google Drives or Edmodo; or public social media network such as Facebook or Twitter.

  Publishing to the NCLab Gallery: submit games to https://nclab.com/turtle-gallery-submit/

  Student work can be viewed at: https://nclab.com/turtle-gallery/

SUGGESTED AGE RANGE FOR STUDENTS AND PREREQUISITES

Tina 2 is designed to teach students between ages 10-18. The younger students tend to progress more slowly but can still be successful. Students at the middle school or high school level generally master the program with little difficulty. Students who have not been exposed to integers, functions, algebraic expressions using variables, and geometry concepts such as ordered pairs, translation and rotation, and solids, may need some background preparation before starting some of the levels.

The main prerequisite for Tina 2 is Tina 1, which covers the basic commands, For loops, and variables. For students with no programming experience, NCLab recommends the Karel courses before taking the Tina course. The Karel courses use a simplified version of Python which is easier to manage.

Students who are in 8th grade or higher will have been taught all the math prerequisites. Students in earlier grades may need math mini-lessons as the course progresses. The math standards and skills are addressed in each Section.

TIME REQUIRED AND SUGGESTIONS FOR COURSE DELIVERY

There are 40 levels or lessons in Tina 2, divided into 5 sections of 8 levels each. The following lessons are written for each section, with notes on the specific skills addressed at each level within the section. Students will naturally slow down as the coding becomes more complex. In general, the amount of time required for the course is about 12 to 18 hours of computer time, including one to two hours for each instructional level, and one hour for the art project level in each section. Here are some suggestions for lesson delivery:

- **As a camp or workshop** that allows long stretches of computer time. Generally, students will complete the Tina 2 course in five sessions of about 3 hours each, although younger or less experienced students will need more time, whereas students who learn easily could complete it in about half that time. Advanced students can experiment with designs, or continue on to one of the other courses.

- **As a self-paced course for independent study**, for computer lab time, after school programs, programming clubs, gifted and talented programs or home study. Students are more likely to complete the course if they are encouraged and supported by adults, and if they have the opportunity to publish and print their designs.

- **As part of an elective computer programming class** at the middle school or high school level. Tina 2 should follow Tina 1. Students who complete both courses will have been introduced to all the basic tools of programming graphic design.

  If taught as a class, time will be spent outside of the computer program to develop and enrich understanding of the underlying geometry and algebra concepts, demonstrate real world applications and share ideas. Students will also appreciate time to build designs on their own. Teachers may break away from programming to teach or review math skills in preparation for a particular Section. The following is a possible 5-day cycle of 50 minute classes for one Section:

Day 1: Introduce concepts and vocabulary (7 minutes)
Watch and discuss YouTube video (8 minutes)
Complete levels 1 and 2 (20 minutes – about 10 minutes each)
Review Levels 1 and 2 as a class (6 minutes)
Assist students with questions while others experiment with designs. (Remaining time)

Day 2: Complete levels 3, 4 and 5 (36 minutes total – about 12 minutes each)
Review Levels 3, 4 and 5 as a class (9 minutes)
Assist students with questions while others experiment with designs. (Remaining time)

Day 3: Complete levels 6 and 7 (30 minutes total – about 15 minutes each.)
Review Levels 6 and 7 as a class (9 minutes)
Assist students with questions while others experiment with designs. (Remaining time)

Day 4: Level 8: Art Project
Review concepts and vocabulary learned (7 minutes)
Introduction: expectations and parameters for project. Distribute checklists (3 minutes)
Plan and Design (15 minutes)
Write and test programs (20 minutes) – note: file can be submitted for course completion and then returned to for further design as a performance task.
Review progress and save files (5 minutes)

Day 5:
Assessment: quiz and/or journal entries. (10 minutes)
Complete Art Project as performance task. (30 minutes)
Share designs (walk around gallery) (10 minutes)

Encourage students who finish quickly to take notes, deepen their understanding of the concepts, and create designs. It is one thing to just complete a level; it is another to truly understand, remember, and apply.

- **As mini-lessons** of about 20-30 minutes each, addressing one to two levels at a time. This might be a good option for upper elementary and middle school where time is a premium. At this rate, the course would take most of a quarter (about 8 weeks) to complete. However, spreading out the lessons may be more successful at reaching students from a broader range of ability and background, because the course is chunked into smaller segments with teacher and peer support.

CROSS-CUTTING CONCEPTS: MATH AND ELA STANDARDS

Tina 2 teaches graphic design, and therefore requires a basic understanding of geometry. At the same time, Tina itself will greatly enhance the teaching of geometry because students can see how geometry affects design. Tina also uses basic algebraic functions to make coding more reliable and flexible. Students will gain an appreciation for the power of algebra.

Geometry Standards

For reference, here is a link to the Common Core Geometry Standards progression from Kindergarten to 12th Grade:

http://www.corestandards.org/Math/Content/G/

It is helpful to refer to this progression when teaching Tina to a particular grade level. For example, 4th grade students are learning about lines, rays, and angles; 5th grade students are introduced to the Cartesian coordinate system; 6th grade students draw polygons in the coordinate system; 8th grade students study the effects of rotation, reflection, translation, and dilation. High school geometry explores curved surfaces and 3-dimensional geometry. Students do not need to know the formulae to write code in Tina 2. Students in earlier grades will need some geometry instruction to prepare them for Tina. An introduction will suffice for most students, as Tina is fairly simple to code. The specific background knowledge needed is referred to in the lesson plans for each section.

Section	Background knowledge required	Standards that apply
1-5 (Tina 1) and 6,7 (Tina 2)	LEVEL 1.1: x,y coordination plane (Cartesian plane); ordered pairs; origin, x and y axes LEVEL 1.1: scale and interval on a coordinate plane LEVEL 1.2: line (line segments), rays and angles LEVEL 1.5 angles in regular polygons, supplementary angles, angles as divisions of 360 degrees. LEVEL 1.7: Integers, using ordered pairs in Quadrants II, III and IV of the coordinate plane LEVEL 4.3: Decomposing angles	4.GA.1,2 and 3 (angles, attributes) 5.GA.1 (x,y coordinate plane) 5.GB.1,2 (attributes of regular polygons) 6.G.A.1 composing and decomposing shapes 6.G.A.3 shapes in the coordinate plane 7.G.A.1 scaled drawings 8.G.A.* translations and rotations, congruent and similar shapes 7.G.B.5 using angle relationships to solve problems.
8	SECTION 8 works with arcs. Although the programs do not require explicit understanding of the formula, students should understand the relationship of arcs to radius.	CCSS.Math.Content.HSG.C.B.5 Derive using similarity the fact that the length of the arc intercepted by an angle is proportional to the radius, and define the radian measure of the angle as the constant of proportionality; derive the formula for the area of a sector

9,10	SECTIONS 9 and 10 work with rotations of 2-dimensional objects.	CCSS.Math.Content.HSG.GMD.B.4 Identify the shapes of two-dimensional cross-sections of three-dimensional objects, and identify three-dimensional objects generated by rotations of two-dimensional objects.

Algebra

Tina makes use of algebraic relationships to write code that is simple and effective. Students apply the properties of operations, use variables as functions, generate patterns based on number relationships, and use order of operations.

The K-5 link for Operations and Algebraic Thinking is

http://www.corestandards.org/Math/Content/OA/ (Operations and Algebraic Thinking)

From 6th grade onward, the study of operations and algebraic thinking is subdivided into different areas of study.

In 6th and 7th grade, the standards focus on ratio and proportion:

http://www.corestandards.org/Math/Content/RP/

In 8th grade, functions:

http://www.corestandards.org/Math/Content/F/

6th through 8th grade, expressions and equations:

http://www.corestandards.org/Math/Content/EE/

As in geometry, students in grades earlier than high school will need some background support in Algebra and Functions to understand what they are doing in Tina.

Section	Background knowledge required	Standards that apply
1-5 (Tina 1)	Use parentheses in numeric expressions. A brief introduction to rational/irrational numbers may help students understand when to use the goto command, and with the tangram project.	5.OA.A1 6.NS.C7 (rational numbers) 8.NS.A1, A2 (irrational numbers)
6	Write a function that describes a relationship between two quantities. Determine an explicit expression, a recursive process, or steps for calculation from a context.	HSF.BF.A.1, 1a
7-10	As above	

Students will also develop good math process skills as they learn to write code. In fact, all of the Common Core Standards for Mathematical Practices apply, so students may very well improve in any regular math studies as a result.

SMP 1: **Make sense of problems and persevere in solving them.** Each lesson is presented as a problem or puzzle to be solved. Students can test their programs instantly, as they go. This feedback encourages them to correct errors and continue until the task is completely solved.

SMP 2: **Reason abstractly and quantitatively.** Students learn how to write logical command sequences, including conditions and repeated routines (loops and nested loops)

SMP 3: **Construct viable arguments and critique the reasoning of others.** In the search for code that meets or beats the criteria, students naturally engage in discussions about the best way to solve a puzzle. They often help each other uncover errors. Class discussions and journals enhance this communication.

SMP 4: **Model with mathematics.** Coding, by its very nature, is translating actions, conditions and goals into defined terms and symbols, which in turn translates into tangible designs.

SMP 5: **Use appropriate tools strategically.** Students have to choose the most effective commands and sequences needed to solve the problem. Subroutines (loops), conditions, and commands are selected to create code that is efficient, robust, readable and flexible.

SMP 6: **Attend to precision.** Programs will not run correctly if there are any logical or syntax errors.

SMP 7: **Look for and make use of structure.** To solve a puzzle, students must break down a task into logical steps.

SMP 8: **Look for and express regularity in repeated reasoning.** Patterns are the key to writing repeated loops, nested loops and conditions.

English Language Arts W.x.10 Write routinely over extended time frames (time for research, reflection, and revision) and shorter time frames (a single sitting or a day or two) for a range of discipline-specific tasks, purposes, and audiences. In Tina, journaling and describing designs help cement learning and give students practice in informational writing.

English Language Arts SL.x. Discussion and collaboration are key ingredients in solving problems, applying learning and creating designs. In particular, students should be able to ask and answer specific questions.

NEXT GENERATION SCIENCE STANDARDS (NGSS)

NGSS is built on three dimensions: Scientific and Engineering Practices (SEP), Disciplinary Core Ideas (DCI), and Cross-Cutting Concepts (CCC).

Learning to write code using Tina develops skills in engineering practices and cross-cutting concepts. Although the performance tasks are described as Art Projects, students may then want to use their modeling and "what if" coding skills to develop models for their studies in the core areas of physical, biological and earth sciences.

Scientific and engineering practices exercised in Tina are

SEP 2: Developing and using models. Since Tina 2 bases program design on functions and variables, students are learning coding habits that will lead them to develop useful testing models.

SEP 5: Using mathematics and computational thinking. Tina 2 helps students make sense of concepts in algebra and geometry. Students create visual representations of equations.

Cross-cutting concepts are valuable tools that can be used to link the skills learned in Tina with fields of scientific and engineering. The main cross-cutting concepts in Tina are

CCC 1: *Patterns. Observed patterns of forms and events guide organization and classification, and they prompt questions about relationships and the factors that influence them.* Students design loops based on patterns. Students can use their skills to build models of repeated patterns found in natural and man-made systems.

CCC 3: *Scale, proportion, and quantity. In considering phenomena, it is critical to recognize what is relevant at different measures of size, time, and energy and to recognize how changes in scale, proportion, or quantity affect a system's structure or performance.* Students learn how to use variables within the functions, which can be used to change scale, proportion and quantity in the main program.

CCC 4: *Systems and system models. Defining the system under study—specifying its boundaries and making explicit a model of that system—provides tools for understanding and testing ideas that are applicable throughout science and engineering.* Students are learning how to create models, which can be then be

CCC 6: *Structure and function. The way in which an object or living thing is shaped and its substructure determine many of its properties and functions.* Shapes can be analyzed in terms of functionality, and refined to improve function.

ETS1, 2, 3: **Engineering Design**

From the NGSS website:

The core idea of engineering design includes three component ideas:

A. Defining and delimiting engineering problems involves stating the problem to be solved as clearly as possible in terms of criteria for success, and constraints or limits.

B. Designing solutions to engineering problems begins with generating a number of different possible solutions, then evaluating potential solutions to see which ones best meet the criteria and constraints of the problem.

C. Optimizing the design solution involves a process in which solutions are systematically tested and refined and the final design is improved by trading off less important features for those that are more important.

How Tina 2 fits this model:

- At the end of each Section, the final Level requires the design of an Art Project. This requires programming and mathematical skills, especially when students are learning how to control the outcome.
- The course sets **criteria and constraints** Students look for the **best solutions** with the simplest, most efficient and robust code.
- Students **test** their designs, learn from failures, and **optimize** their code, just as they would in real life engineering tasks.
- Once students develop some proficiency, they can **apply** their skills to an unlimited range of projects.

To view the Engineering Design in the NGSS document in detail:

http://www.nextgenscience.org/sites/ngss/files/Appendix%20I%20-%20Engineering%20Design%20in%20NGSS%20-%20FINAL_V2.pdf

VOCABULARY, LANGUAGE AND PROGRAM SUPPORTS

- **Instruction:** Text complexity (Lexile score) ranges from about 440 to 800L, suitable for 3rd to 4th grade upwards.
- **YouTube videos:** demonstrate steps learned in the lesson. Links are listed within the lessons.
- **Text size** can be adjusted for readability.
- **Design viewer:** Students can view their designs in the right hand block at any time, providing instant visual feedback.
- **Error message feedback:** If the program contains errors, the line and error type are flagged with comments.
- **Vocabulary:** words that are specific to programming, math terms, and any Tier II or III words that students may encounter are listed and described under each Section in these lesson plans.
- **Student Journal:** a journal is provided for concept and vocabulary review, and reflections on learning. It includes sketch pages to design programs while offline.

BACKGROUND-BUILDING AND SUPPORT ACTIVITIES

Art, Science, and Math are all rich in geometric patterns that can inspire design. Hands on activities in any of these areas help students make sense of computer graphics and abstract lines of code.

Art projects:

- Op art drawings, tessellations and origami develop skills with shapes, lines and angles.
- Pottery wheels, lathes and drills can be used to create rotational shapes and shells.
- Woven, knitted, crocheted, tie-dyed, quilted and beaded designs demonstrate repeated patterns in fabric arts.

Art appreciation: Geometry has been used throughout human history as a component of art.

- Mosaic patterns range from simple to complex, especially in later examples of Islamic art.
- Rules of composition, including the golden mean.
- Abstract art, especially art that is based on optical illusion.

Patterns in Science: Geometric patterns abound in nature: x-ray diffractions of atomic structures, seashells, the rotating arms of distant galaxies.

Patterns in Engineering: Cloverleaf-shaped highway interchanges, gears.

Geometry and Algebra: Instruction and review of geometry and algebra concepts (see lesson plans for specific skills).

Ruler and protractor work: physically drawing shapes, lines and angles. Specifically, learning the different ways to divide up 360 degrees into equal parts.

Geometric manipulatives: tangrams, two and three-dimensional shapes, linking rods and connectors such as K-nex, fraction block sets (circles, polygons, rods, tiles), all build physical and visual awareness of geometry. Tesselations are examples of nested loops.

Realia. Collect items that show patterns and repeated patterns. Students can bring items to make a tabletop or bulletin board display.

Examples

- Stamp sets (stamps have designs; designs can be stamped into a repeated pattern);
- Fabric items (braid, lace, knitted or woven patterns)
- Natural items such as seashells
- Manufactured items such as chains, bracelets, gears

- An oscilloscope or oscilloscope simulation app can be used to show wave patterns.

Student Journal Sharing. Journaling provides an opportunity to reflect on learning and deepen understanding of concepts and procedures. It is a place to imagine new designs and programs. All of this can be shared as partners, small groups or whole class.

Failure is an Option. After students have passed a level, have them change a line in their program that would make it fail. Rotate the students to a different work station. Can they find the error? This is a great team exercise.

Hour of Code (https://code.org/learn): As a warm-up to Tina, students can benefit by exploring free Drag and Drop programming games found at Hour of Code.

DEPTH OF KNOWLEDGE

Most problems in the Tina 2 course have more than one possible solution given the parameters, although there are less solutions that fit the number of lines required. The object of the lessons is to write code that is clean and simple and to avoid redundancies that can lead to errors. As a result, the Depth of Knowledge (DOK) during the instructional phase is 1 to 3. Using the creativity suite, DOK 3 and 4 level problems can be created and solved.

BLOOM'S TAXONOMY

Application and Analysis: Students must analyze the given parameters to come up with a solution. Students immediately apply what they are learning at each stage by writing a program.

Synthesis and Creation: Students can create their own publishable designs, bringing together all the skills they have learned.

ENRICHMENT, REMEDIATION AND PROGRESS MONITORING

Since the course is self-paced, students can move through the lessons based on their own rate of learning.

Steps can be repeated at any time for review and reinforcement.

Teachers should monitor and provide support as needed. At some point, most students will hit their own personal threshold level in which they aren't immediately successful. Point out the built-in examples, hints and line prompts within the program. Follow up with discussions about what they learned from revisiting these supports.

In a camp or workshop setting, it is important to build in physical breaks. Students tend to stay longer than they should in front of the computer.

In any setting, encourage opportunities to interact and discuss progress.

Set design challenges for students, especially for those who go through the levels quickly.

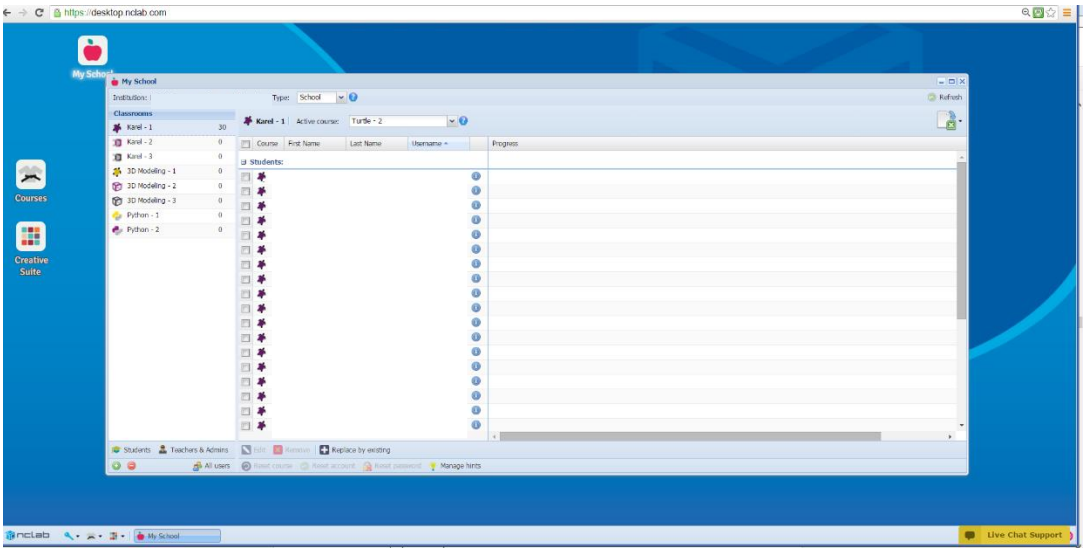
ASSESSMENT

Each section includes focus questions that can be used to guide a classroom discussion, or answered as a short essay or quiz after students complete the section.

Formative assessment is built into each level. Students get immediate feedback from the program. The design created by their code is displayed to the right, and any syntax errors are noted in the dialogue box below their code.

Upon successful completion of a level, students will unlock the next level. Likewise, upon successful completion of each section, students will receive a certificate and unlock the next section.

Teachers can monitor the progress of their students by clicking on the My School apple. This opens a new window. Select Turtle-2 to Active Course.



Journals: A Student Journal is included in the course materials and can be used as a portfolio artifact.

Note: Journals are available as a separate document from NCLab.

Art projects as performance assessments: The final level in each section is an Art Project, in which students write a program that creates a geometric design based on the skills learned in that section. In the Lesson Plans, a printable assignment is included at the end of each section that can be used to score the designs.

Student Feedback: At the end of each level, students are asked to evaluate the level of difficulty by clicking on an EASY, MEDIUM or HARD button. Students can add specific comments. This gives the NCLab designers valuable feedback for improving the games.

Here is a possible grading scheme that includes several methods of assessment:

SUGGESTED GRADING SCHEME FOR TINA 2			
Type of Assessment	**Points**	**Number**	**Total Points**
Section Completion	**50**	**5**	**250**
End of Section Art Project	**50**	**5**	**250**
Journal Responses	**50**	**5**	**250**
TOTAL POINTS			750

LESSONS

Note: The best way to prepare for these lessons is to do them as a user either ahead of time or alongside the students. Answer keys are available, and will be emailed to teachers with the course subscription.

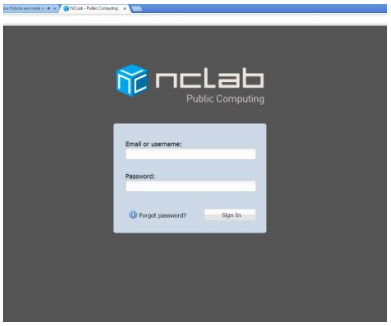
INTRODUCTION TO THE COURSE (ABOUT 20-30 MINUTES)

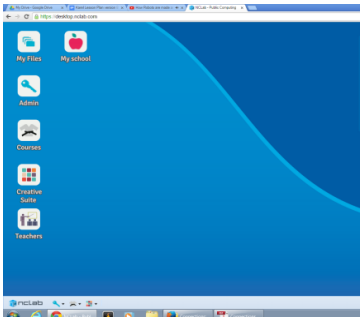
Students should already be familiar with logging into the course. If needed, review the procedures for logging in and starting the course. Demonstrate the log in steps and first lesson on a computer (for larger classes, attached to a projector or Smartboard if available).

Modeling How to Access the Course:

Model how to log in, select the course from the Desktop and navigate screens with initial level, using Smartboard or projector. Have the students complete each step as you go along.

Log in to account https://desktop.nclab.com/.
Select "Courses" (double-click)

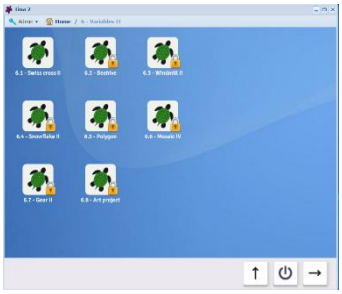




Select Tina Turtle and then Tina 2.

Select the Section (6-Variables II) and Level (6.1. – Swiss Cross II)

Note that only one section and level are available. The rest are locked until the section or level is successfully completed.



SECTION 6 VARIABLES II: LEVELS 6.1 – 6.8

Objectives: Students will be able to write and use parametric variables in their programs.

Vocabulary:

For reference, here is a list of basic commands and definitions learned in Tina 1, presented in order of appearance in the levels:

`tina.go(n)`, where `n` = the number of steps.

This command moves the turtle forward, creating a line on the graph

It can also be written as

`tina.forward(n)`

`tina.fd(n)`

`tina.left(n)`, where `n` = the number of degrees to turn.

`tina.right(n)`, where `n` = the number of degrees to turn.

These commands will turn the turtle by the number of degrees indicated.

Left can also be written as `tina.lt(n)`, and right as `tina.rt(n)`

`tina.pu() or penup();tina.pd() or pendown()`

Pen up (so that Tina moves forward without drawing a line) and pen down (to resume drawing with forward movement):

`tina.width(n)`, sets the line width, where `n` is the line width in units.

`tina.hide()` hides the turtle so that it doesn't show in the final drawing

`tina.back(d)`, where `d` is the distance Tina backs up. Tina will not draw when backing up.

`tina.extrude(n)`, where `n` is the width to be extruded. Extrude sets the width of the shape in the z axis so that the design can be printed. Extrude also hides Tina, so the hide command does not need to be used if the extrude command is used.

`tina.goto(x,y)`. Tina will draw a line to a specified coordinate pair. This is a useful command for angles that are not integers or decimals to the nearest tenth. It should only be used if necessary.

Loop: Loop: a set of repeated commands. There are two different types of loops in Python: The counting **(for) loop** which repeats the set of commands a given number of times, and the conditional **(while) loop** which repeats the set of commands while a given condition is satisfied (the number of repetitions does not have to be known in advance). Tina only uses the for-loop. The set of commands that will be repeated is called the "body of the loop".

Algorithm: a series of logical steps that leads to the solution of a task. Students may be familiar with algorithms used in operations such as subtraction and long division.

Logical error: a mistake in an algorithm. Planning helps reduce the number of errors.

Computer Program: An algorithm written using a programming language.

Syntax: the way a command line is written.

Syntax error: a mistake in spelling, operators, indentations, spaces

Nested loops: a nested loop is a repeated pattern or loop, which is part of another repeated pattern, or loop. For example: if I want to make a row of 10 triangles, I can write a loop for to make a triangle, and nest it inside a loop that will draw that triangle ten times in a row.

Variable: in terms of programming, variable is the name and value of something that will be recorded in memory. In the For loop, the i is an **index or counting variable**. If we set i to a range of values, then i will change each time the loop starts the body of the program. If we then use i as part of a command, that command will output a different value each time.

Range: the range is the lower and upper limit of the variable i. Note that if the range is set with only one value, then the lower limit of the range is assumed to be 0, with the number in the parentheses being the upper limit. Important: the final value will be the difference between the upper and lower limit. For example, in range (1,11), the final value used in the program is 10, or 11-1.

New vocabulary for Section 6:

Parametric variable: a variable that specifies a parameter, such as the length of a side. A line is written ahead of the For loop, in which the parametric variable equals a value. The parametric variable is then used within the loop instead of a number.

Time required:

Once students have watched the course overview and settled into Tina 2, the first three levels can be completed quickly, since students are only adding or changing a line or two. After that, students are doing more of the programming, and the length of time will vary based on experience and ability. Levels 6.6 and 6.7 require some thought, planning, and trial-and-error. The Art Project in Level 6.8 and other responses such as quizzes, journals, and discussion can be expanded or shortened to suit the course.

Oftentimes, the pacing concern is slowing students down so that they absorb and retain what they have learned. Actual programming time will vary from about one to two hours.

Background knowledge/Introductory Set/Purpose:

Since students will have already completed Tina 1, it may not be necessary to provide an overview of the course. On the other hand, this may be an opportunity to review progress and look forward to the remaining lessons. The following link shows students an overview of Tina 2 (which starts with Basics and ends with Variables) and Tina 2 (which starts with Parametric Variables and ends with Rotational Solids and Shells). Here is the link to the course overview:

https://nclab.com/turtle-course-details/

Share a couple of examples of design in art and nature. A quick Internet search will produce many pictures and videos to serve as examples.

Purpose: Section 6 (Levels 6.1-6.8) teaches parametric variables, which allow students to define variables which are then used in the program. The variables can be assigned different values without rewriting the code in the program itself. This information is on the help cards for Section 6, which can be introduced now, reviewed as students work through the Levels, and summarized at the end of the Section.

Math background, Tina 1: students may need extra instruction depending on their background in math. This can be explicitly taught, reviewed, or assigned as determined by the teacher. Students should already be familiar with the following concepts used in Tina 1:

- *x,y* coordination plane (Cartesian plane); ordered pairs; origin, *x* and *y* axes
- Scale and interval on a graph
- Line (line segments), rays and angles
- Angles in regular polygons, supplementary angles, angles as divisions of 360 degrees.
- Integers, using ordered pairs in Quadrants II, III and IV of the coordinate plane

Math background, Tina 6.1: Students should all have a rudimentary understanding of variables and functions, which are taught in some form from elementary school onward. By high school, students are learning more formal representations of functions as expressions and sentences containing independent and dependent variables.

Direct instruction:

As previously recommended, it is helpful to have two teacher accounts: one for learning the course ahead of the students, and one for modeling steps to the students. This second account is useful for instruction as it will show the screens as they appear for the first time to a user.

Since Tina is designed to be a self-paced course, the amount of direct modeling should be kept to a minimum. The course itself teaches new commands and concepts. It is helpful to introduce each Section and perhaps work through the first Level together.

Teaching points are suggested in each level and can be used as mini-lessons.

The class may benefit from watching and discussing the YouTube videos together. There are currently no videos for Section 6. In Tina 2, the videos start in Section 7.

Help cards are included in each section. These can be printed and distributed or displayed on a bulletin board. The help cards are useful for modeling and remembering command syntax and purpose. Basic command help cards are found in the lesson plans for Tina 1.

Here is the help card for Section 6:

Using Parametric Variables (Tina 2 Level 6)

A parametric variable is a variable based on one or more parameters used in the program. For example, parameters can be line length, spacing, or number of lines.

The parametric variable is defined and assigned a value before the program. Then, the variable name is used within the program instead of values. This way, the value can easily be changed without rewriting the program itself.

Example (Swiss Cross):

```
s=30
for i in range(4):
  tina.go(s)
  tina.left(90)
  tina.go(s)
  tina.right(90)
  tina.go(s)
  tina.left(90)
```

s is the length of the side.
Within the program, s is used with the go(s) command.

To make a different sized cross, assign a different value to s on the first line.

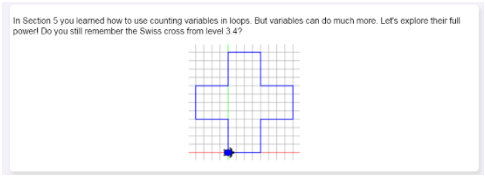
Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced instruction: Levels 6.1-6.8

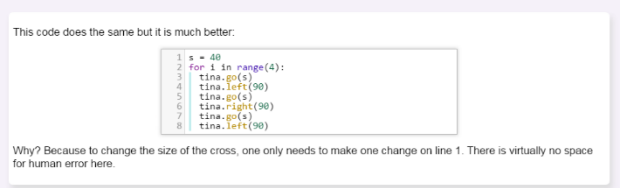
6.1 Swiss Cross II

The first screen introduces the section by reminding students of a pattern that they designed in 3.4, the Swiss Cross.

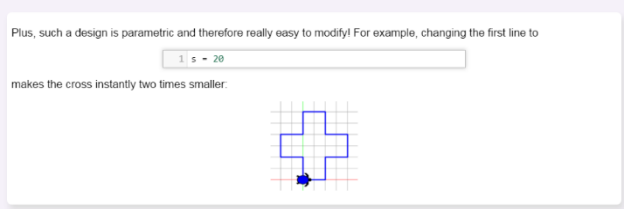


In the original level, the code was written like this. The length of each side was specified in the code for that line.

In the improved code, the length is defined ahead of the For loop: s = 40. Then, within the loop, the variable is used to specify the lengths. This is a cleaner, less error-prone approach.

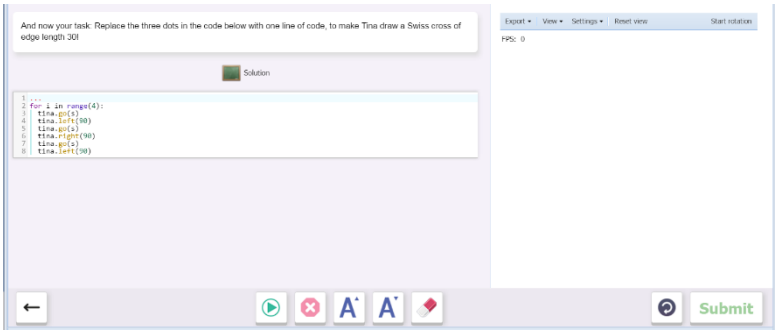


If a new length is needed, we just change the value of s instead of changing each line of programming. S = 20 produces a cross that is half the size of the original.



Complete the program by adding a variable at the beginning, and using it within the program. The required length is 30 units.

Students will add s=30 to the program. The program will produce a Swiss cross in the right screen of length 30.

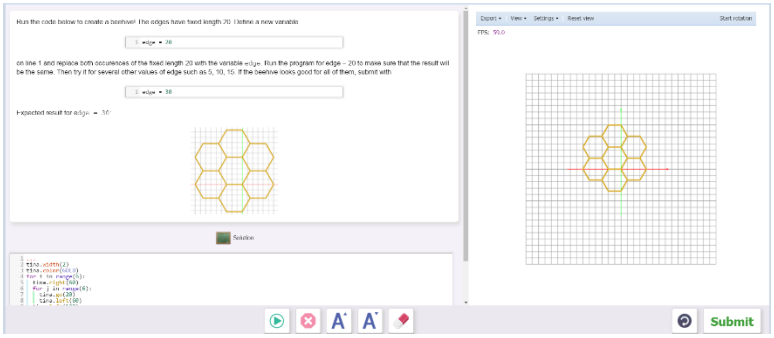


6.2 Beehive

Add the parametric variable `edge` that specifies the lengths of the sides to the beginning of the program.

Begin with edge=20. Replace the `go(20)` lines in the program with `go(edge)`

Then try some other values for `edge`: 5, 10, and 15. Finish with 30 and submit.

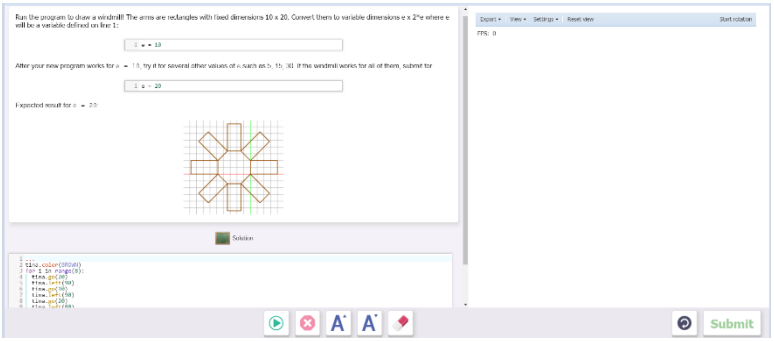


6.3 Windmill II

Practice using a parametric variable `e` to draw the windmill.

The length of the rectangle is two times the width, so 2*`e` will be used for the length, and `e` will be used for the width. Replace the values in program with `e` or 2*`e` as appropriate.

Run the program with different values for e. Submit the program with e=20.

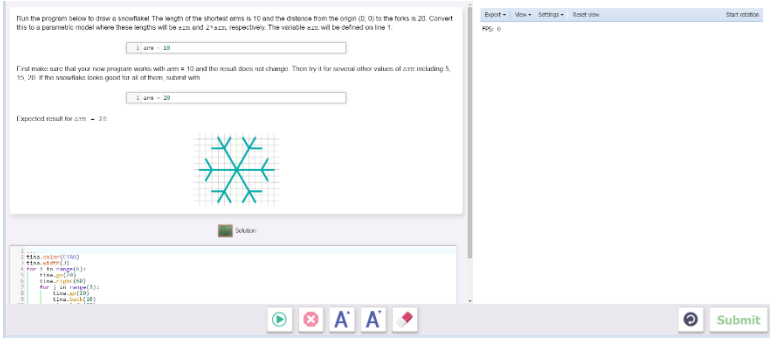


6.4 Snowflake II

Practice using a parametric variable `arm` to draw a snowflake.

The long portion of the arm is twice as long as the short portion, so again, the variable can be used as arm and 2*arm within the program.

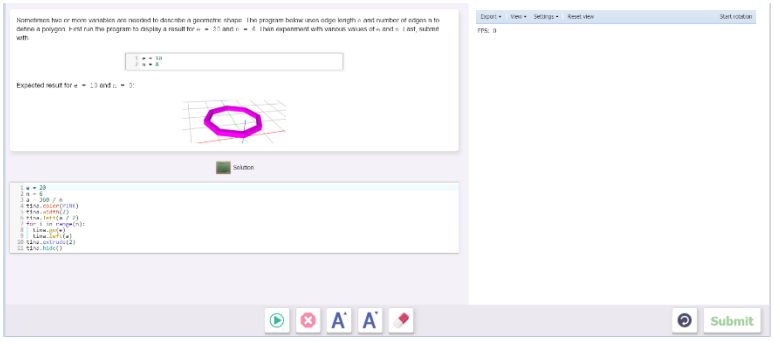
Run the program with different values for arm. Submit the program with arm=20.



6.5 Polygon

Practice using multiple parametric variables to create a polygon.

This program uses **three** parametric variables: use e to specify the edge length, n to specify the number of sides, and a=360/n to specify the angle (notice that a is derived from n)

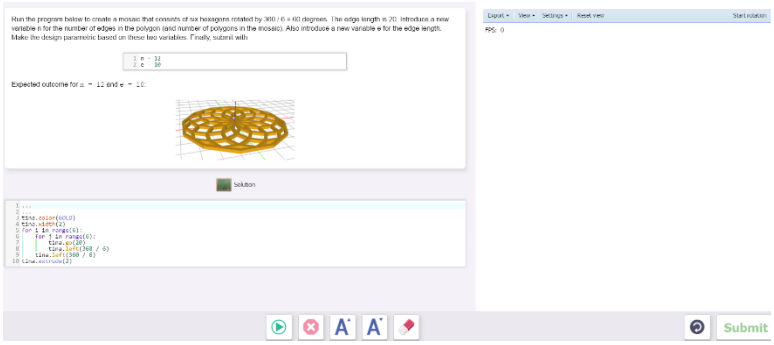


Run the program using different values. Submit the program with e=10 and n=8.

6.6 Mosaic IV

Practice using multiple parametric variables to create a mosaic.

Specify a variable `n` for the number of edges in the polygon, and `e` for the length of the edge.



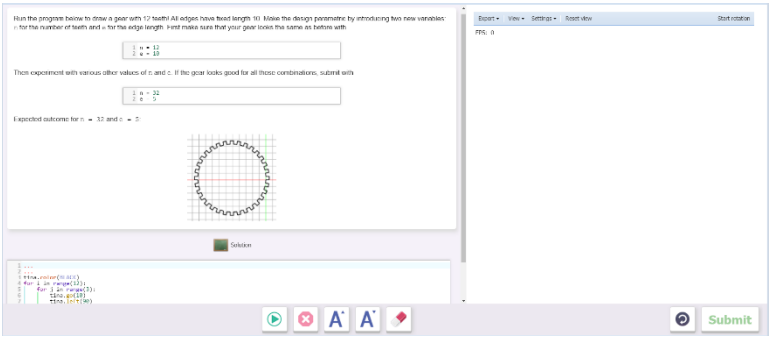
Edit the program by replacing values with the variables where appropriate.

Run the program using different values. Submit the program with n=12 and e=10.

6.7 Gear II

Practice using multiple parametric variables to create a gear.

Specify a variable `n` for the number of teeth and `e` for the edge length.



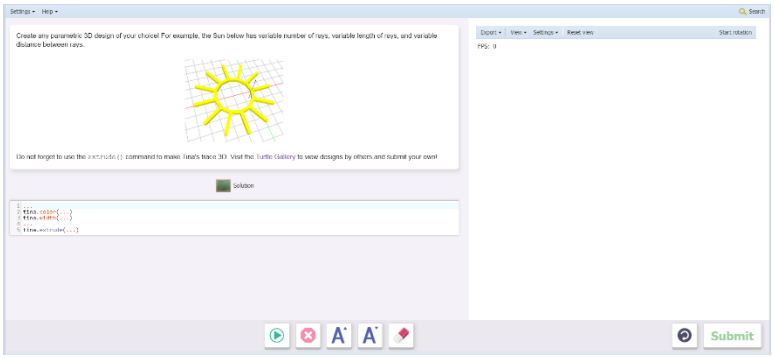
Edit the program by replacing values with the variables where appropriate. Run the program using different values. Submit the program with `n`=32, `e`=5.

6.8 Art Project

Create any parametric design of your choice.

The example of the Sun design has variable number of rays, length of rays, and distance between rays.

Use the `extrude()` command to make the design printable.

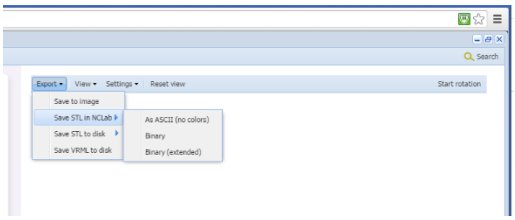


As in Tina 1, students create a 3-dimensional shape by extruding the 2-dimensional shape (z-axis). The line width must also be thickened in order to make a printable design. The design can be rotated on the screen to view it in from all angles.

Possible commands: `go(), left(), right(), back(), pu() or penup(), pd() or pendown(), home()`

Number of lines: will vary with design.

The file can be saved through the Settings drop-down menu. Subscribed users have access to storage on the NCLab server. Save as an STL file for 3D printing.



The next screen will show a Yellow Belt of the Fourth Degree certificate, which can be saved or printed.

This project is the final level for Section 6 and can be used as an assessment.

Suggested focus questions for post-session discussion and written responses. (10-20 minutes):

What advantages are there to writing parametric? (By using parametric variables, the actual values can be set before the program without rewriting the program itself. This makes the program very flexible and less error prone.)

How could you use parametric variables to test designs in art, crafts, trades, engineering, or science? (Try different patterns, shapes; find limitations and exceptions; find best fit)

When do you use parametric variables, and when do you just assign a fixed value? (That would depend on the application. Do you want to control some variables and experiment with others?)

Sharing programming experiences:

How many lines of programming did your project need to create your design? Which values did you decide to change to parametric variables? How did the design change with different values?

Assessment:

Within the program itself, students receive a Yellow Belt of Fourth Degree certificate upon successful completion of Section 6. The certificate can be printed, emailed or shared on social media.

Students Journal entries can be used as assessment.

The Art Project can be used as a performance task or portfolio artifact. Here is a possible 50-point assignment card:

Section 6 Assignment

END OF SECTION 6: ART PROJECT (50 POINTS)

Objective: Create and publish a design based on parametric variables.

Planning (15 points): After you have decided on a design, draw out it on grid paper or in your journal. In this section, limit your design to straight line segments. Curves come later.

- Find a design or sketch one of your own.
- What patterns do you see?
- What parameters of the design can be assigned to parametric variables (length, spacing, number)?
- Are patterns made up of smaller patterns (nested loops)
- What will you name the overall design?
- When will the `pu()`, `pd()`, `back` and `home` commands be useful?

Writing the code (25 points): Write the program.

- Use comment lines to describe different segments of the program.
- Write the code and test as you go.
- Include examples of some or all of the previously learned commands, such as `go()`, `left()`, `right()`, `back()`, `pu()` or `penup()`, `pd()` or `pendown()` and `home()`

Saving the file, sharing and publishing (10 points)

- Publish the design to your folder. Inform someone else about the game by providing the link on _____, or by email.

Support:

For students who need extra support:

- Show them the next step needed.
- Use the printable help cards.
- Partner them with a more experienced student.
- Decrease the number of functions required for the Art Project.
- Cut and paste a function from an earlier lesson. Have student modify the function to make something new.

Enrichment: Creative Suite

For students who are ready to create more designs, Turtle Tina is available in Creative Suite.

- Students can create their own files.
- Creative Suite can be accessed by double-clicking the Creative Suite button on the left side of the NCLab desktop, or from the pullup menu at the bottom of the screen.
- On the Creative Suite menu, click on Programming, then Turtle Tina.
- The opening screen includes a YouTube video and sample code. Students can run the code and experiment with it.
- The video and a List of Commands are always available from the Help menu.
- To create a new file, select New from the File menu.
- The file is saved in the same way as the Art Project, to the NCLab Folder.

SECTION 7 FUNCTIONS: LEVELS 7.1-7.8

Objectives: Students will be able to write functions. In Section 7, students will learn how functions can be defined, then used within the main program.

Vocabulary:

Function: a function is a set of commands written separately from the program. Instead of using specific numbers, the function writes the commands using **variables**. The main part of the program will call the function and specify the values for the variables. The function is given a name.

`Def`: the `def` command is used to define a function. For example:

`def` polygon (T,e,n): means "Define a function named polygon for turtle T, length of edge e, number of sides n."

The turtle used in the commands must always be included in the parentheses. Other variables are defined as needed.

Time required:

The first three levels can be completed quickly, since students are only adding or changing a line or two. After that, students are doing more of the programming, and the length of time will vary based on experience and ability. Levels 7.6 and 7.7 require some thought, planning, and trial-and-error. The Art Project in Level 7.8 and other responses such as quizzes, journals, and discussion can be expanded or shortened to suit the course. Oftentimes, the pacing concern is slowing students down so that they absorb and retain what they have learned. Actual programming time will vary from about one to three hours.

Background knowledge/Introductory Set/Purpose:

Section 6 taught how to use parametric variables, which make it easy to change the values without disturbing the code in the main program. Functions take this a step further. A function can define a set of commands using variables that can be called in the main program. This time, the value of the variables is assigned in the main program, which is located after the function rather than before it.

Purpose: Section 7 (Levels 7.1-7.8) teaches functions, which allow students to write sections of code that can be called up by the program. This information is on the help cards for Section 7, which can be introduced now, reviewed as students work through the Levels, and summarized at the end of the Section.

Math background, Tina 7.1: Students should all have a rudimentary understanding of variables and functions, which are taught in some form from elementary school onward. By high school, students are learning more formal representations of functions as expressions and sentences containing independent and dependent variables.

Direct instruction:

Since Tina is designed to be a self-paced course, the amount of direct modeling should be kept to a minimum. The course itself teaches new commands and concepts. It is helpful to introduce each Section and perhaps work through the first Level together.

Teaching points are suggested in each level and can be used as mini-lessons.

The class may benefit from watching and discussing the YouTube videos together. For Section 7, the videos are located in Levels 7.1 and 7.4 in this document and in the course itself.

Here are the help cards for Section 7:

Writing a function using the `def` command (Tina 2 Level 7)

The def command is written on the first line. It includes the name of the function followed by parentheses, which include a letter for the name of the turtle and any variables. It ends with a colon, just like a For loop.

The body of commands for the function are indented two spaces. The body can include single commands and loops.

Example:

```
def polygon (T, e, n):
  for i in range(n):
    T.go(e)
    T.left(360/n)
```

Polygon is the name of the function
T is the name of the turtle Tina
e is the length of the edge
n is the number of sides, so it defines both the range and the angle to be turned.

Calling a function in the main program (Tina 2 Level 7)

The function has to be called in the main program. This is also where the variables are assigned numeric values.

Example:

polygon (tina, 20, 5)

This assigns
tina to the T,
20 to the e, making an edge length of 20 units
5 to the range, which will create 5 sides
And also 5 to create the left turn angle (360/5)

The main program can assign different values to the variables to make different polygons

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Levels 7.1-7.8

7.1 Stop Sign

The opening screen provides a link to a video that teaches Functions. This video can be viewed on YouTube at

https://www.youtube.com/watch?v=xiUaYio78z8&feature=youtu.be

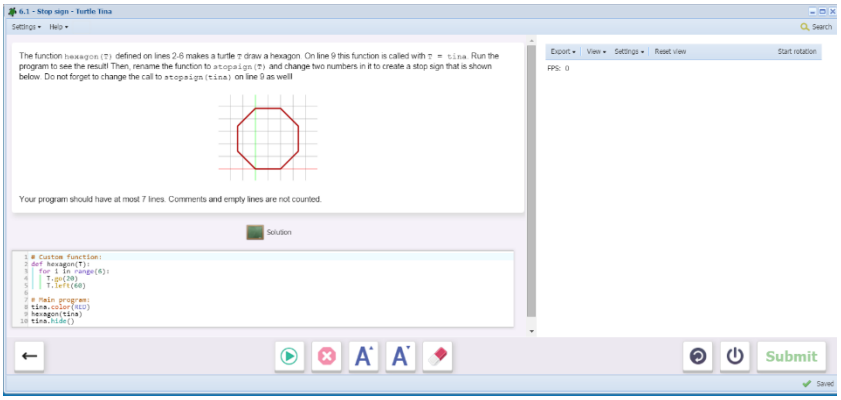
If students do not have access to You Tube, the video can be shown to the class. Run time is 5:58.

Discuss the value of creating functions. Students will see that they can write a function that allows them to create different shapes by selecting different variables for the same function.

In Stop Sign, students create a function called `stopsign` by overwriting a function called hexagon. A stopsign is, of course, an octagon.

A model of what the design should look like is included in the upper left pane, along with any specific instructions such as the commands to be used and the number of lines in the program.

Geometry hint: the turn angle is equal to 360 degrees divided by the number of sides. For 8 sides, the turn angle is 360/8 = 45 degrees.



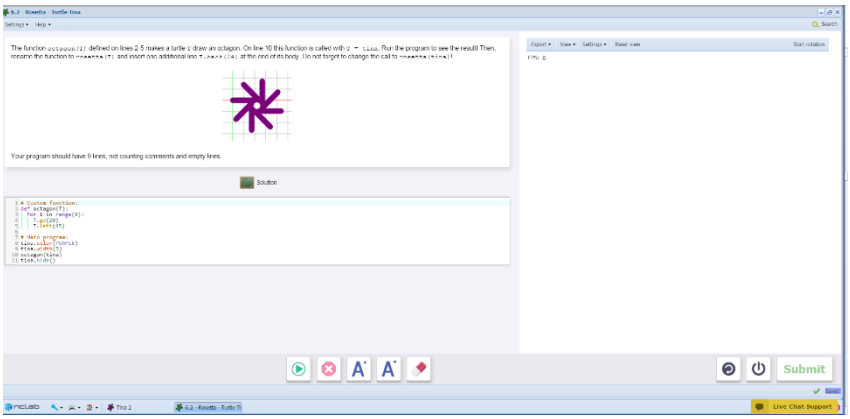
After modifying the program, students can preview the results by pressing the green play button at the bottom of the screen.

Once the stop sign looks like the one in in the preview screen, students press the green Submit button. A green message box will appear if the program is successful. The green message box lists the successful task or tasks, and may contain additional pointers.

It will also ask students to rate the level as too easy, just right, or too hard and provide any comments. This feedback helps the program developers make improvements.

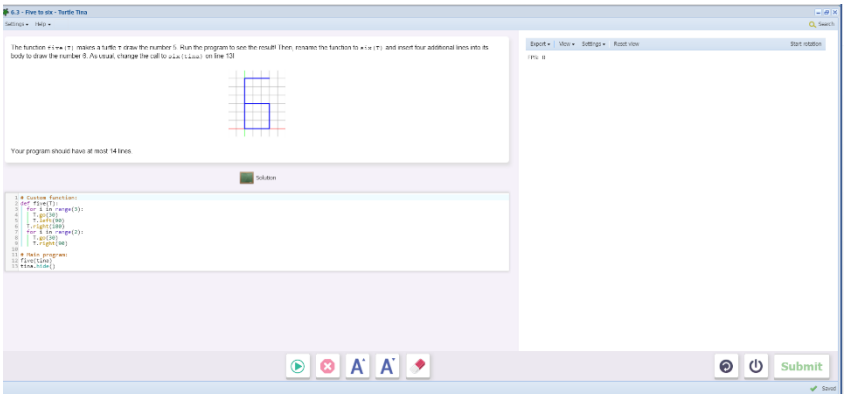
7.2 Rosetta

In Rosetta, students convert an octagon to a radiating shape. This time they overwrite the function names, and insert an additional line `T.back(24)`



7.3 Five to Six

In this level, students learn to write a function to make the number 6 by changing a program that makes the number 5.



Commands:

`go(),left(),right(), back()`

Number of lines: 14 (student adds 4 lines)

The instructions remind students to change the program name to six(tina).

There is more than one way to create the 6. Encourage students to compare their solutions. Which one is the simplest? (The answer key inserts the 4 lines before the first loop).

7.4 Vortex

Vortex builds a figure similar to the one that was demonstrated in the Variables video in Tina 1. The program is a little different than the one used in the video, but the concept is the same. The 7.4 vortex is created by drawing a line, turning 89 degrees, and then **decreasing** the line length by 1 each time.

Here is a link to the Variables video:

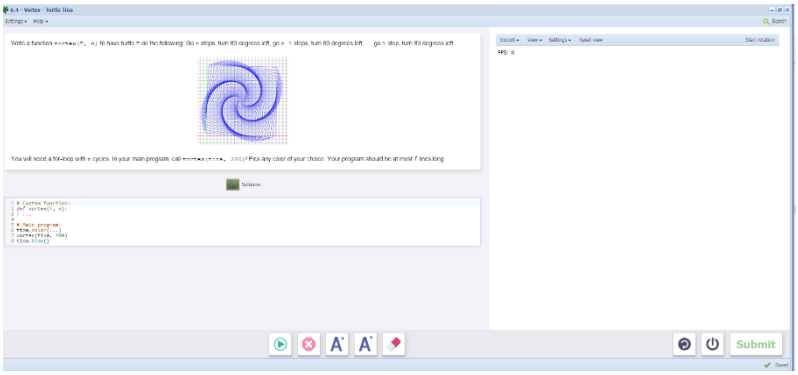
https://www.youtube.com/watch?v=tJXRgo4M0qg

Students write the function commands by creating a FOR loop. A color is chosen at the end. The program runs the function Vortex 200 times.

Commands:

`go(), left(),`

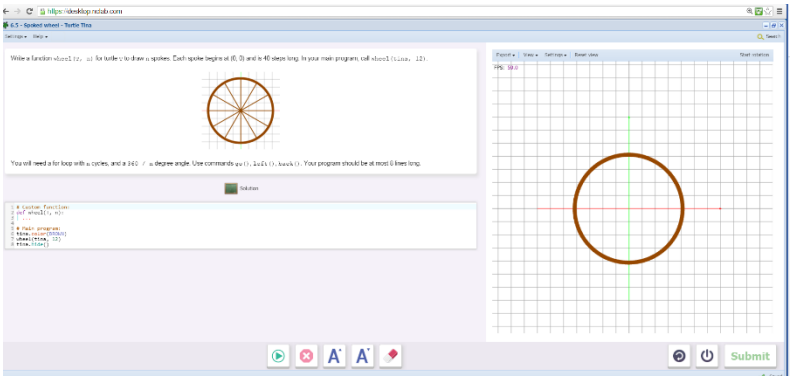
Number of lines: 7



This program takes time to run because of all the iterations. Students may want to play with the angle and the number of iterations to see what happens. Why does it make sense to stop at 200 iterations?

7.5 Spoked Wheel

In this level, students write a function that will draw the spokes of the wheel. This is another opportunity to see the value of writing a function. The variable n equals the number of lines, and the turning angle equals 360/n. To get a specific number of spokes, we **call** that number in the main program rather than the function. Students can try calling different values to see how this works.



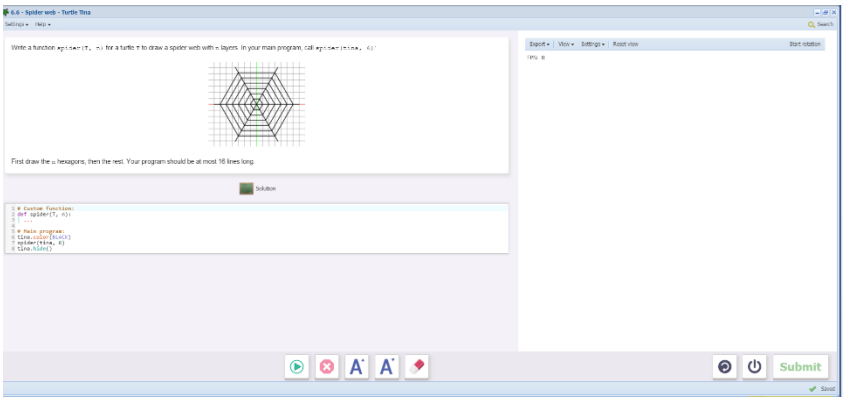
To make a spoke, a line is drawn, then the back command brings Tina back to origin, and a left command turns Tina, ready to start the next line.

Commands: `go(), left(), back()`

Number of lines: 8

7.6 Spider Web

In Spider Web, students create a much longer program, divided into two sections. Students should write comment lines as headings for each of these two sections: one for the hexagons, and one for the spokes (referred to as #The Rest in the solutions manual).



For the hexagons, students create two functions: the function i to position Tina at the start of a loop, and the function j to draw the hexagon. The second function loop is nested within the first one. The starting position and the lengths of the sides of the hexagons will change with i.

Reminder: Each grid square equals 10 units, so the distance can be defined as 10*i.

Once the hexagons are drawn, we need a second set of commands to draw the spokes (after the second comment line #). Use the `home()` command to get Tina back to origin at the start of each spoke. Because the spokes extend 10 units beyond the hexagons, 10 is added to the length.

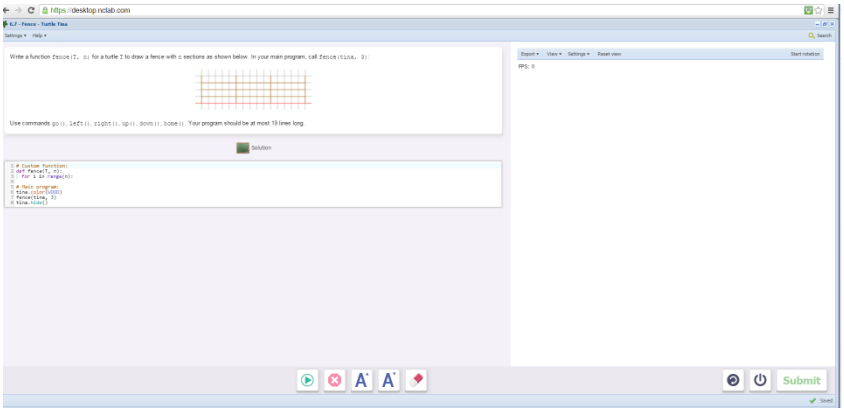
Commands: `go(), left(), right(), back(), Home()`

Number of lines: 16

Challenge students to replace 6 and 60 with variables n and 360/n. Does the program run correctly? (It should) Does it create a similar figure for other values of n? (There are problems) When should we specify the values and when should we write the functions using variables?

7.7 Fence

In Fence, students create two functions: one to make the poles (the verticals) and one for the logs (the horizontals). The fence requires 4 poles and 3 logs.



In addition to the back command, the pen up and pen down commands are used to move Tina to the different starting positions without drawing a line.

Commands: `go(), left(), right(), back(), pu() or penup(), pd() or pendown(), home()`

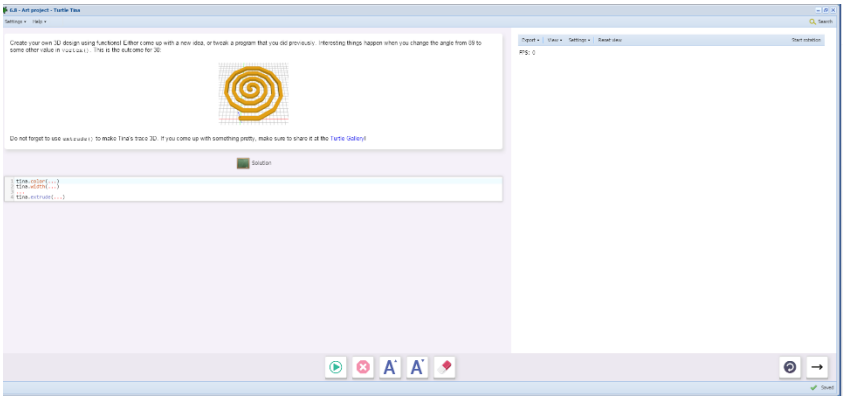
Number of lines: 19

Students can count off the number of grid squares to determine the lengths and spacing. The number of poles is one more than the number of logs (n+1). The length of the logs is a multiple of the number of logs and can be written as such (n*50). Use the home command to return Tina to the starting position after drawing the poles (verticals).

7.8 Art Project

In this level, students will create a 3D design based on a function. Students can modify one of the functions they have already learned, or create a new function from scratch.

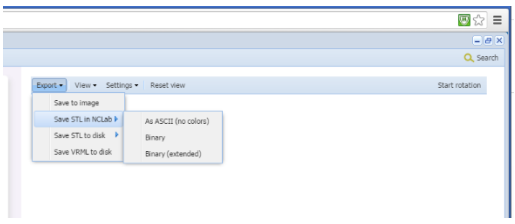
As in Tina 1, students create a 3-dimensional shape by extruding the 2-dimensional shape (z-axis). The line width must also be thickened in order to make a printable design. The design can be rotated on the screen to view it in from all angles.



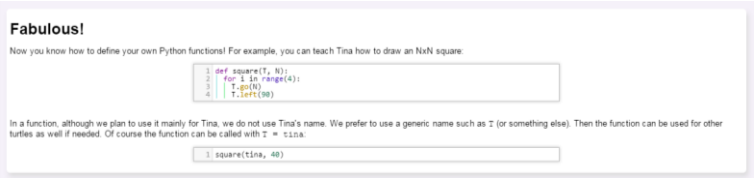
Possible commands: `go(), left(), right(), back(), pu() or penup(), pd() or pendown(), home()`

Number of lines: will vary with design.

The file can be saved through the Settings drop-down menu. Subscribed users have access to storage on the NCLab server. Save as an STL file for 3D printing.



After submitting their projects, students will see this screen, congratulating them on being able to write functions.



The next screen will show a Purple Belt of the First Degree certificate, which can be saved or printed.

This project is the final level for Section 7 and can be used as an assessment.

Suggested focus questions for post-session discussion and written responses. (10-20 minutes):

What advantages are there to writing functions? (By using variables, the actual values can be called from the main program without rewriting the function. This makes the program very flexible and less error prone.)

How could you use functions to test designs in art, crafts, trades, engineering, or science? (Try different patterns, shapes; find limitations and exceptions; find best fit)

When do you use variables, and when do you just assign a fixed value? (That would depend on the application. Do you want to control some variables and experiment with others? Does the function work for the whole range of values that you would like to use?)

Sharing programming experiences:

How many lines of programming did your project need to create your initials? How did you organize the program (dividing tasks into functions and main program)? How many functions did you use?

Assessment:

Within the program itself, students receive a Purple Belt of First Degree certificate upon successful completion of Section 7. The certificate can be printed, emailed or shared on social media.

Students Journal entries can be used as assessment.

The Art Project can be used as a performance task or portfolio artifact. Here is a possible 50-point assignment card:

Section 7 Assignment

END OF SECTION 7: ART PROJECT (50 POINTS)

Objective: Create and publish a design based on functions.

Planning (15 points): After you have decided on a design, draw out it on grid paper or in your journal. In this section, limit your design to straight line segments. Curves come later.

- Find a design or sketch one of your own.
- What patterns do you see?
- Can the patterns be broken down into separate functions (think about the spider web, made up of a set of hexagons and a set of radiating lines)
- Are patterns made up of smaller patterns (nested loops)
- What will you name the overall design? What will you name the functions?
- When will the `pu()`, `pd()`, `back` and `home` commands be useful?

Writing the code (25 points): Write the program.

- Before you write the code for each function, write a comment line that names and describes the shape.
- Write the code and test as you go.
- Include examples of some or all the commands learned in Section 1: `go()`, `left()`, `right()`, `back()`, `pu()` or `penup()`, `pd()` or `pendown()` and `home()`

Saving the file, sharing and publishing (10 points)

- Publish the design to your folder. Inform someone else about the game by providing the link on _____, or by email.

SECTION 8 ARCS: LEVELS 8.1-8.8

Objectives: In this section, students learn to use the `arc` command.

Vocabulary:

Arc: part of the circumference of a circle, or a curve.

Arc command: The command is written as `tina.arc(a, b)`, where *a* is the number of degrees, and *b* is the length of the radius.

A third parameter can be included, telling Tina which direction to turn when writing the arc:

`tina.arc(a, b, 'r')` tells Tina to turn right. Note the single quotation marks.

`tina.arc(a, b, 'l')` tells Tina to turn left

An instructional video on writing arcs is included at the beginning of Level 8.1.

Prerequisite skills: Completion of Section 7, and an understanding of all previous commands. Section 8 assumes a basic understanding of the geometry of circles: radius, arc, the number of degrees in a circle.

Time required: Time required will vary based on student ability and experience. Most students will complete this section in two or three hours, excepting the art project.

Background knowledge/Introductory Set/Purpose:

Use photographs such as those suggested in Background Activities to demonstrate Celtic designs, which are based on arc geometry. Arcs are also used in engineering and architecture: not only are they beautiful, they are also strong. Examples are bridges, doorways, highway interchanges.

Arcs are curves based on the circle.

To define an arc, we need to know the number of degrees we will turn and the radius.

To start the arc in the right direction, we need to orient Tina before writing the arc command. We do this by turning Tina left or right by a number of degrees.

We can also tell Tina which direction to draw, either left or right.

Direct Instruction and Modeling:

Review how arcs are based on a circle. They are a portion of the circumference defined by the radius and angle of turn. Play the video in 8.1. The first level may be modeled to the class. Here is the help card for Section 8.

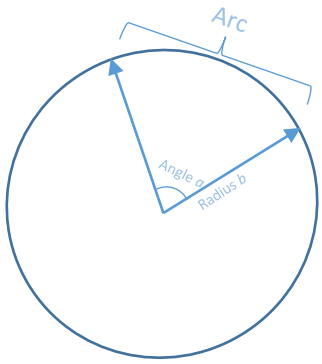
Writing an Arc command (Tina 2, Section 8)

Arc command: The command is written as tina.`arc` (a, b), where a is the number of degrees, and b is the length of the radius.

A third parameter can be included, telling Tina which direction to turn when writing the arc:

tina.arc (a, b, 'r') tells Tina to turn right. Note the single quotation marks.

tina.arc (a, b, 'l') tells Tina to turn left



Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 8.1-8.8

8.1 Triquetra

Triquetra starts with a You Tube video explaining how to draw arcs. The video can be viewed from within the program or by following this link:

https://www.youtube.com/watch?v=tcbCM0So9MY&feature=youtu.be

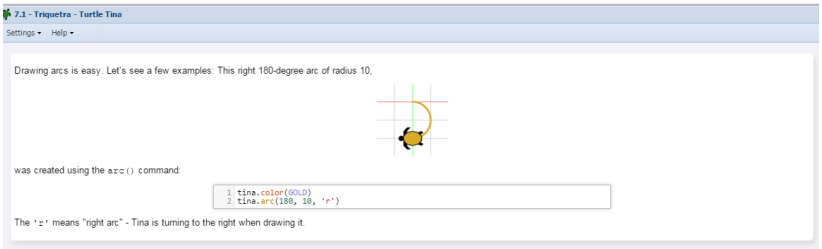
The next screen demonstrates the code required to draw a simple arc. Only one line of code is needed.

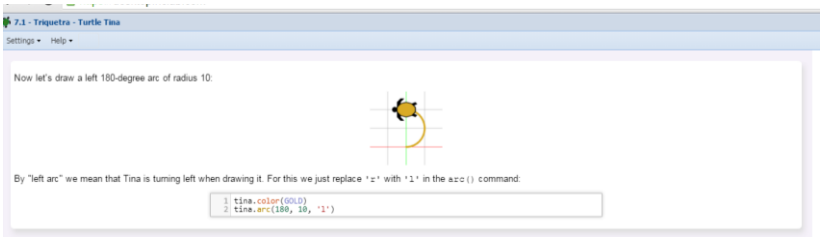
```
tina.arc(180, 10, 'r')
```

or

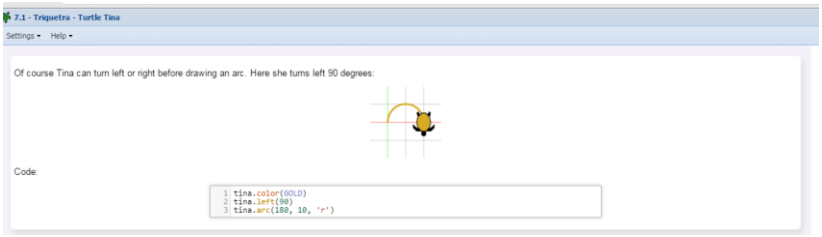
```
tina.arc(180, 10, 'l')
```

depending on whether Tina is drawing to the right or left.

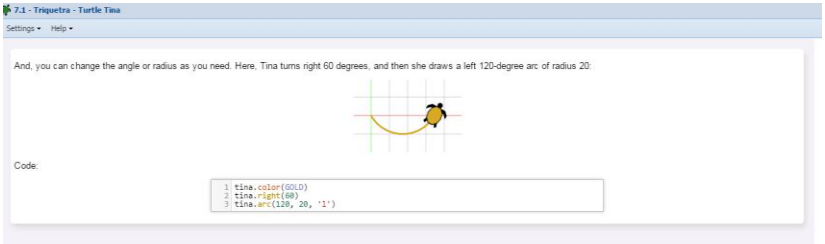




Choosing which direction to draw Tina is not the same as orienting her starting position. To get Tina oriented, a left or right command precedes the arc command.



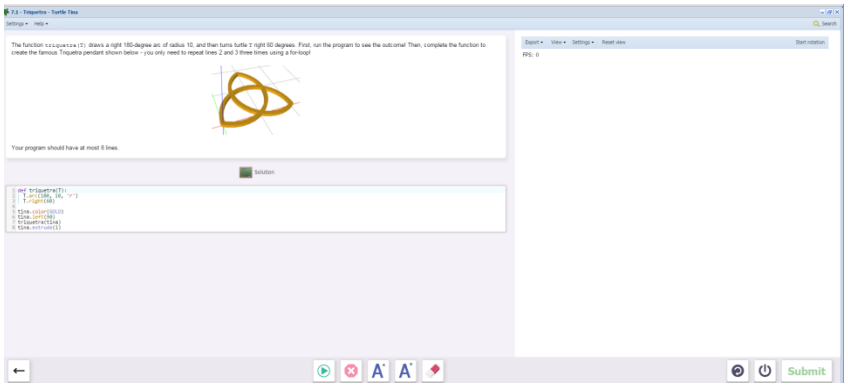
The next screen demonstrates how to control the arc by changing the number of degrees to draw and the radius.



In 8.1, students practice creating an arc within a function, by adding the for loop. The arc command and other lines are already written.

Number of lines: 8

Commands: `go()`, `left()`, `right()`, `arc()`



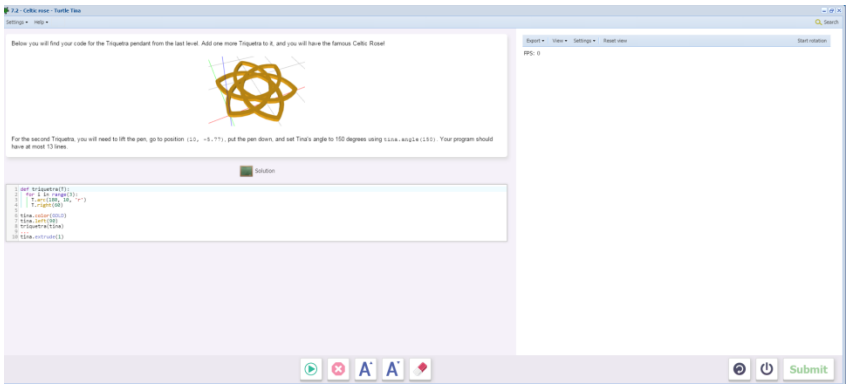
8.2 Celtic Rose

Students use the triquetra function from 7.1 to create a celtic rose. To do this, a second triquetra overlaps the first.

Specific instructions are given in the upper left panel: “For the second Triquetra, you will need to lift the pen, go to position (10, -5.77), put the pen down, and set Tina's angle to 150 degrees using `tina.angle(150)`.” Students will write these commands in the main program, which calls the triquetra function twice.

Number of lines: 13

Commands: `go()`, `left()`, `right()`, `pu()` or `penup()`, `pd()` or `pendown()`, `angle()`, `arc()`

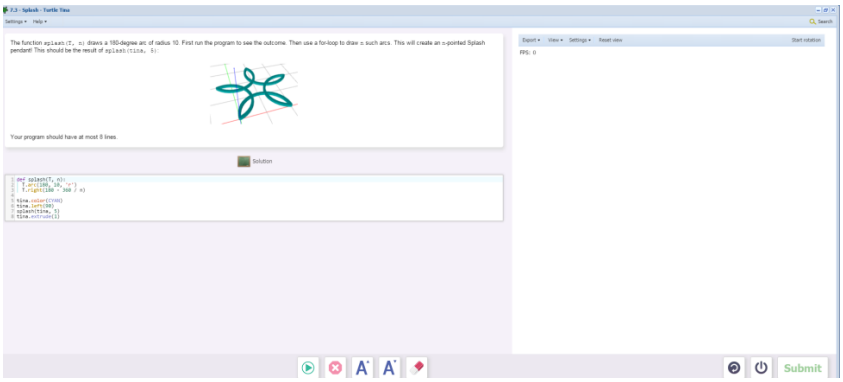


8.3 Splash

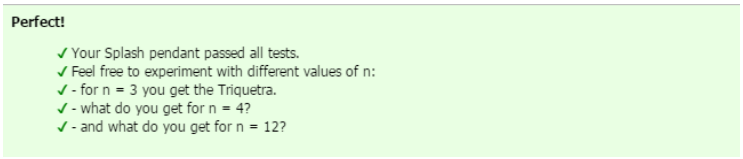
Students use the arc command to make a shape named Splash, by adding the For command. The number of points on the shape is defined by n in the For Command. The value of n, 5, is called in the main program.

Number of lines: 8

Commands: `go()`, `left()`, `right()`, `arc()`

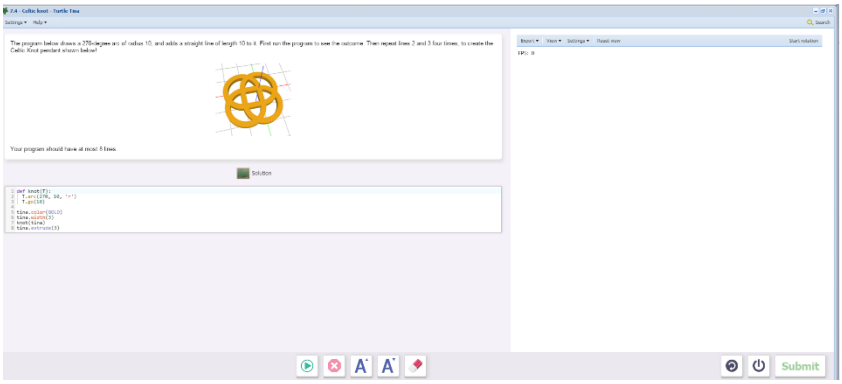


After submitting the design, students will see this note, encouraging them to try different values for n.



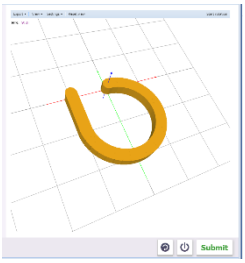
8.4 Celtic Knot

Students draw a celtic knot by combining an arc and a straight line, and then repeating that pattern to form the knot.



The instruction box suggests running the program to see the arc/line combination before adding lines. The insert shows what one arc/line looks like.

Students add a For loop to repeat this sequence 4 times.



The arc being used is 270 degrees, a multiple of 90. Students might want to try other multiples of 90 (including those above 360) to see what happens.

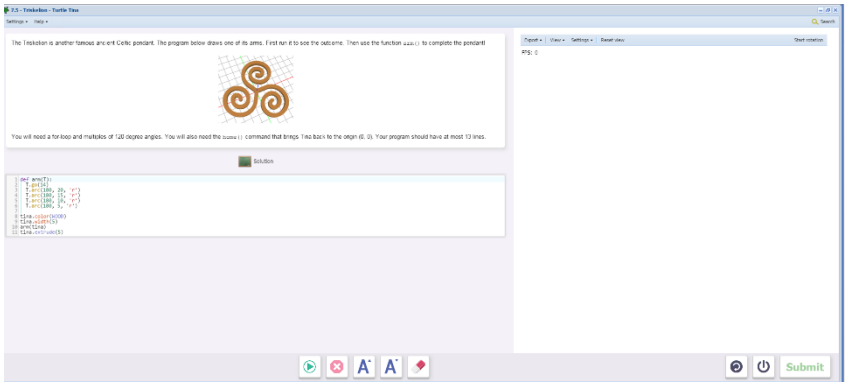
Number of lines: 8

Commands: `go()`, `arc()`

8.5 Triskelion

Students write code that will repeat a coil three times to form a triskelion design.

The coils are made up of a set of arcs that decrease in length by 5 units each time, always drawing to right ('r'). Although the function for the coil is premade, it is worth discussing the code with the students, so that they will know how to do it for future projects. As in 7.4, students should run the program before adding lines to one coil.



Within the main program,

Tina is turned to start the coil. The turn angle is a multiple of 120 degrees. This can be written as a For loop, multiplying 120 degrees by i each time.

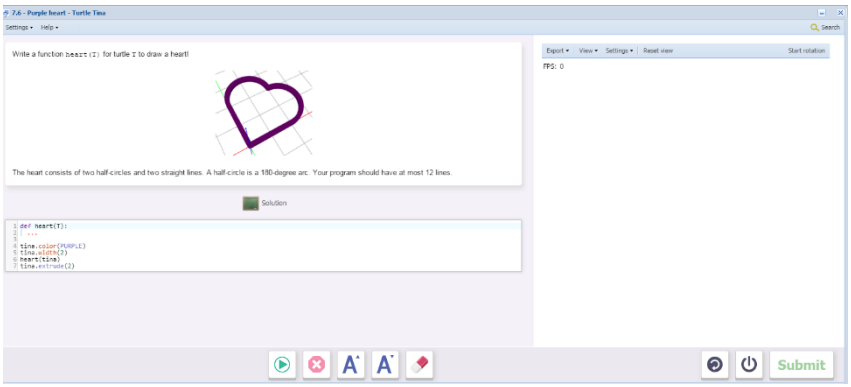
After finishing a coil, Tina returns home (to 0,0) to start the next coil.

Number of lines: 13

Commands: `go(), left(), home(), arc()`

8.6 Purple Heart

Students write a function that draws a heart. This time students write the function, including arc commands.



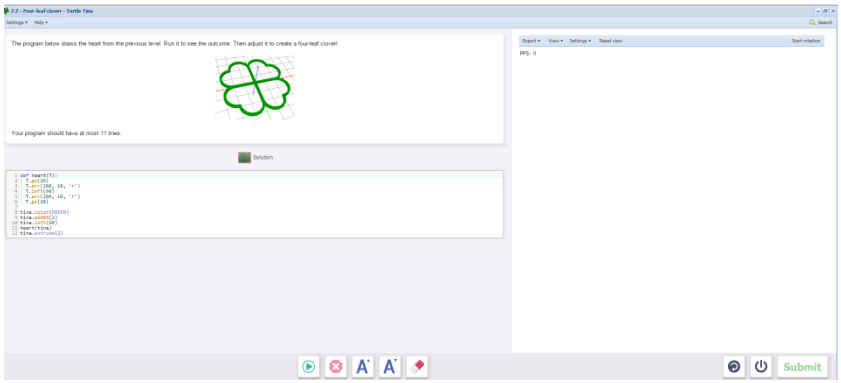
The suggested solution draws the function as line, arc, arc, line – a simple 4 line program.

Number of lines: 12

Commands: `go(), left(), arc()`

8.7 Four-leaf Clover

Students write a program to make a four-leaf clover, based on 4 repetitions of the heart function written in 8.6.



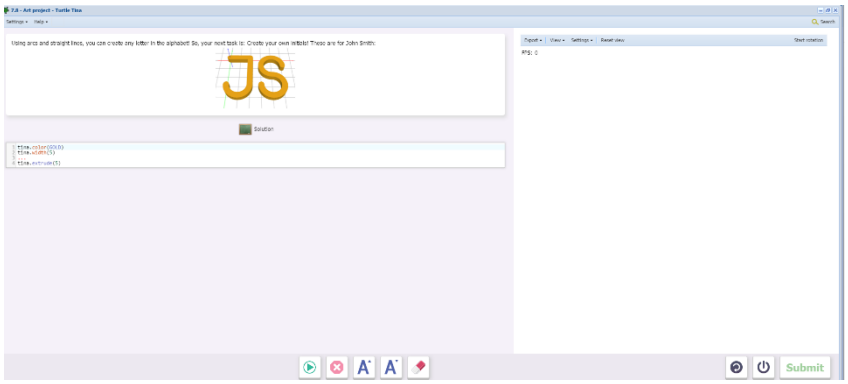
The heart function is already written. Students can test run the code to see the single iteration before inserting the For loop in the program.

Number of lines: 11

Commands: `go(), left(), arc()`

8.8 Art Project

Students will create their initials as the art project for Section 8. A sample for the letters J and S is shown in the instruction box.



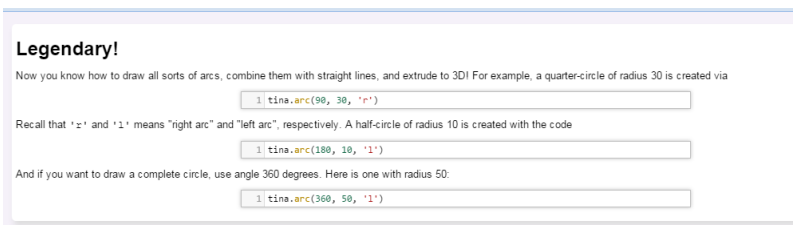
Since this section is about arcs, students should find a way to include curves in their design. Not all capital letters contain curves. Students can substitute lower case letters or their own font design.

To continue practicing functions, students should create each letter as a function, then call it within the program.

The number of lines and commands used will vary. In addition to the functions, the main program will need commands to place Tina at the start of each letter.

The figure must be extruded in order to print.

After submitting their Art Project, students will see this message on the following screen, summarizing what they have learned in Section 8:



Legendary!

Now you know how to draw all sorts of arcs, combine them with straight lines, and extrude to 3D! For example, a quarter-circle of radius 30 is created via

```
tina.arc(90, 30, 'r')
```

Recall that `'r'` and `'l'` means "right arc" and "left arc", respectively. A half-circle of radius 10 is created with the code

```
tina.arc(180, 10, 'l')
```

And if you want to draw a complete circle, use angle 360 degrees. Here is one with radius 50:

```
tina.arc(360, 50, 'l')
```

The next screen will show a Purple Belt of the Second Degree certificate, which can be saved or printed.

This project is the final level for Section 8 and can be used as an assessment.

Focus questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

What three components are needed to describe an arc? (number of degrees, radius, right or left)

Can you visualize a portion of a circumference? How do you decide what values to use? (Visualizing helps to estimate the radius and turning angle needed)

Can arcs as used in Section 8 be used to draw any curve? (No. Simple arcs can be drawn based on circles or ellipses, but other curves need more complex geometry. Students may have attempted some of these in their art project and been frustrated.)

Sharing programming experiences:

How many lines of programming did your project need to create your initials? How did you organize the program (dividing tasks into functions and main program)? How many arcs did you use? Did you run into any problems designing the arcs?

Assessment:

Within the program itself, students receive a Purple Belt of Second Degree certificate upon successful completion of Section 8. The certificate can be printed, emailed or shared on social media.

Students Journal entries can be used as assessment.

The Art Project can be used as a performance task or portfolio artifact. Here is a possible 50-point assignment card:

Section 8 Art Project

SECTION 8 ART PROJECT: CREATE A SET OF INITIALS (50 POINTS)

Objective: Create and publish a set of initials in Tina Turtle. The initials must include arcs, so you may need to make lower case letters, or create a font that uses curves.

Planning (15 points): After you have decided on a design, draw out the letters on grid paper or in your journal.

- Describe each letter. What commands will be needed to draw them? Name each shape (for example, Letterj) and describe the color, starting position, arcs, line lengths, and turning angles.
- Each letter will be a function that is called in the main program.
- Write out the order in which you will program the letters, and any pu()/pd() or movement commands in between the letters.

Writing the code (25 points): Write the program.

- Before you write the code for each letter function, write a comment line that names and describes the letter.
- Write the code and test as you go.
- Use some or all of the commands learned in Section 1: `go()`, `left()`, `right()`, `arc()`, `back()`, `pu()` or `penup()`, `pd()` or `pendown()` and `home()`
- Remember to set the width, to extrude the shape, and to hide Tina at the end.

Saving the file, sharing and publishing (10 points)

- Publish the design to your folder. Inform someone else about the game by providing the link on _____, or by email.

SECTION 9 SOLIDS: LEVELS 9.1-9.8

Objectives: In this section, students learn to create rotational 3D solids. Students start by drawing a set of lines or arcs along the y-axis, which is the axis of rotation, then using the command `rosol()` to turn the line into a 3D shape.

Vocabulary:

Rotational solid: a solid created by rotating a trace around an axis. The trace can be composed of line segments, arcs or combinations of these.

rosol: a command written as tina.`rosol`(), which rotates a line trace about the y-axis to create a 3-dimensional solid. It is written after the commands used to create the line trace. Think of the distance of the line from the y-axis as a radius.

Time required:

Time required will vary based on student ability and experience. Most students will complete this section in two hours.

Prerequisite skills: Completion of Section 8.

Background knowledge/Introductory Set/Purpose:

A real life example of rotating a line to make a solid shape is using a wood lathe to turn cylinders of wood into elaborate posts, table legs and candlesticks. The lathe spins the wood around an axis, and the carver carves the shape along an imaginary line.

3D shapes are used to create working models of everything from toys to cars.

Purpose: Section 9 (Levels 9.1-9.8) teaches how to build 3-dimensional solids by rotating a line around the y-axis.

Direct Instruction and Modeling:

Show the videos in 9.1 on a projector. https://youtu.be/1lLBEj_T_xc

Discuss any questions that arise. Model or guide the first lesson (9.1), based on student needs.

The only new command is rosol. Otherwise, there are no new commands to be learned in this section. Additional instruction may be needed, and this is specifically addressed in each level. The help card is on the next page.

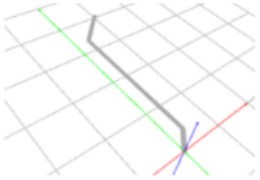
Individual/Group practice: The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Writing a Rosol command to create a rotational solid (Tina 2, Section 9)

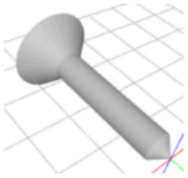
Rosol command: The command is written as `rosol()`

Create a set of commands that will draw a line trace. Follow this set with the rosol command to rotate the shape around the y axis.

Example:



```
tina.color(STEEL)
tina.left(45)
tina.go(5)
tina.left(45)
tina.go(30)
tina.right(45)
tina.go(10)
tina.rosol()
```



Self-paced Instruction: Levels 9.1-9.8

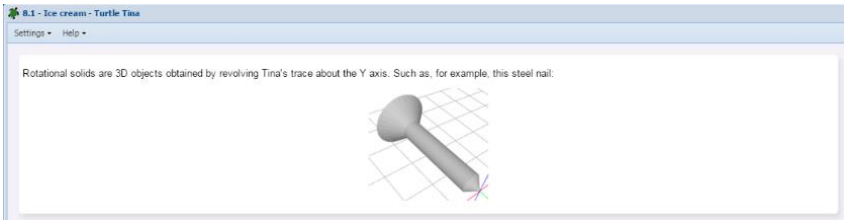
9.1 Ice Cream

9.1 begins by showing a video about 3D symmetrical solids. The video can be called up from within the program, or by following this link:

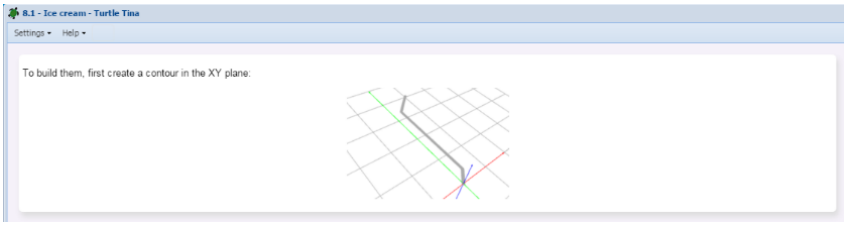
https://youtu.be/1lLBEj_T_xc

The next set of screens teach how to use the `rosol()` command, step by step.

The first screen defines rotational solids. The example shown is a nail.



The next screen shows the trace that will be rotated to make the nail.

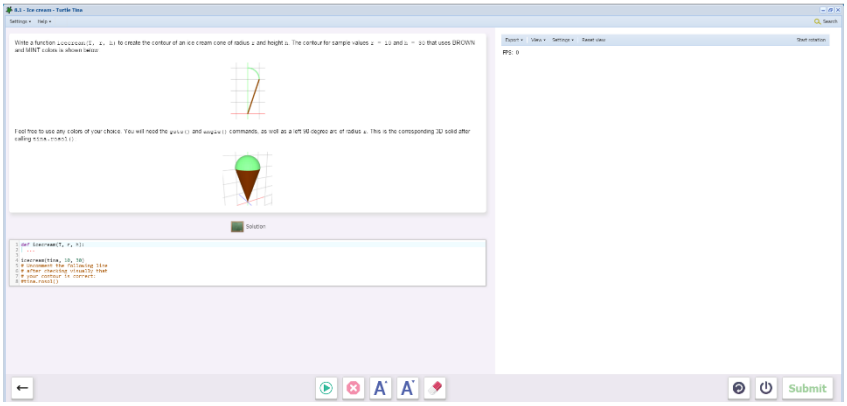


The third screen shows the lines of programming, including the command `rosol()` at the end, which will rotate the trace.

In 9.1, students create an ice cream cone. The trace is simply composed of one line and one arc.

The `goto()` command is used to write the line to the (10,30) coordinate position.

The `angle()` command is used to reset Tina's position before drawing the arc.



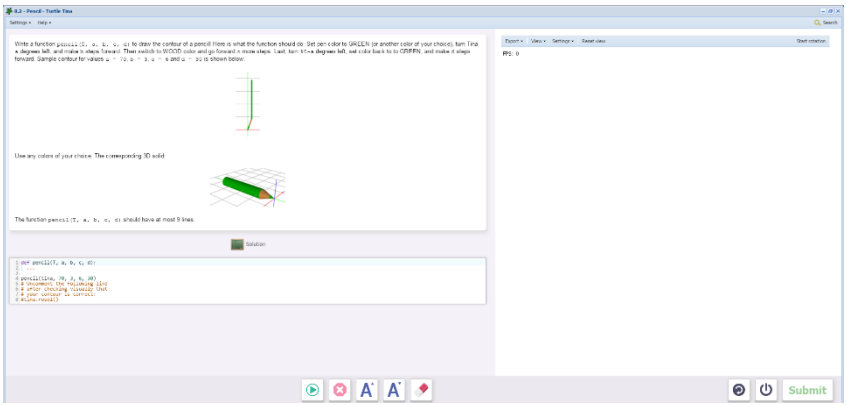
Commands: `goto()`, `angle()`, `arc()`, `color()`, `rosol()`

Number of lines: 8

9.2 Pencil

Students create a pencil by drawing the trace, then `rosol()` to create the solid.

The preferred solution uses variables within the function, then assigns values to each variable when the function is called in the main program. Note that the point of the pencil is on the origin, so the cone is drawn first.



The function is pencil (T, a, b, c, d), where

- a = degrees turned left
- b = length of the lead (cone)
- c = length of the wood (cone)
- d=the length of the cylindrical portion of the pencil.

The variables and their role in the function are explained in the instruction box.

Subtraction is used to compute one of the angles (90 - a). This difference relationship is often used, especially when the whole is fixed (in this case, the whole is 90 degrees), and the sum of the variables equals the whole.

Students can experiment with the colors in all of these designs. They are not limited to the suggested colors.

Note that the `rosol()` command has a comment # in front of it. Students are encouraged to run the trace first, correct any errors in the function, then erase the comment symbol # in front of the `rosol()` command to complete the program and create the solid.

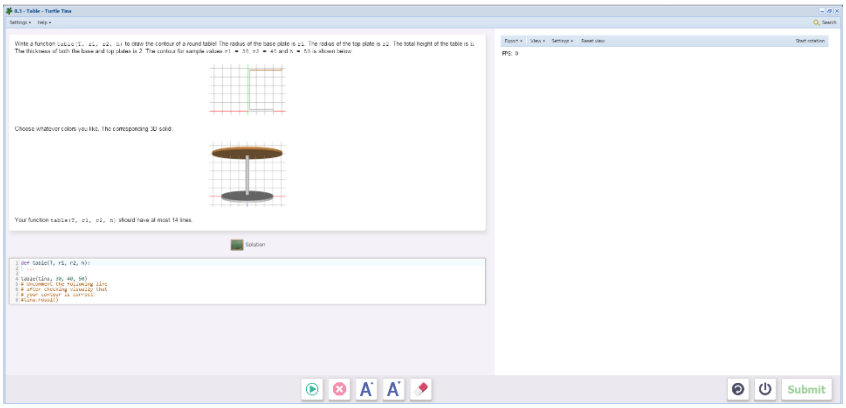
Number of lines: 9

Commands: `go()`, `left()`, `color()`, `rosol()`

9.3 Table

Students practice writing a function to use with `rosol()` by creating a round, pedestal table. Detailed explanations are given in the instruction box.

The variables in Table have conventional names: r1 for the radius of the base, h for the total height, and r2 for the radius of the top. The thickness of the top and base is 2, so some dimensions will include subtracting 2 to account for thickness.



Number of lines in the function **table**: 14

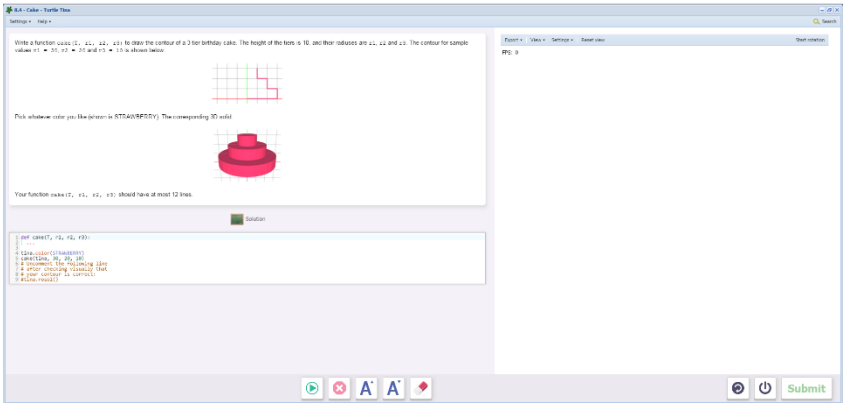
Function Commands: `go(), left(), right(), color()`

Main program commands: `rosol()`

9.4 Cake

Students practice writing a function to use with `rosol()` by creating a 3-tiered cake.

Variables for the three radii are r1, r2, and r3. The height of each layer is 10. As in Table, subtraction is used to compute some distances.



Number of lines in the function **cake**: 8

Function Commands: `go(), left(), right()`

Main program commands: `color(), rosol()`

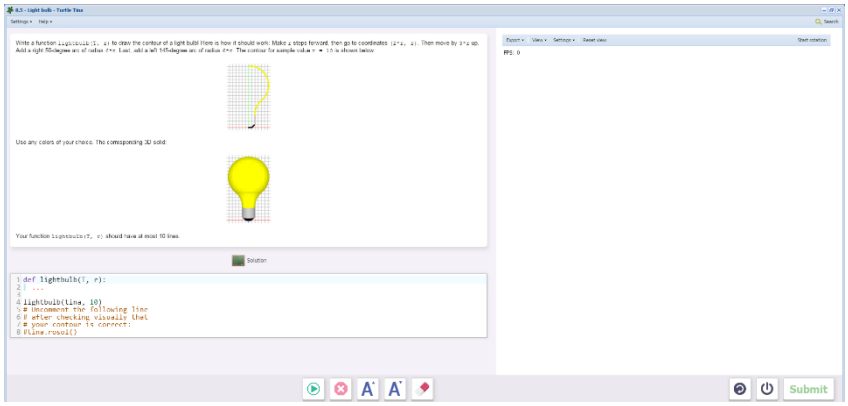
9.5 Light Bulb

Students practice writing a function to use with `rosol()` by creating a light bulb. The light bulb, of course, is a curved surface and will require the `arc()` command. The function also uses goto(). Here are the screen instructions:

"Write a function `lightbulb(T, r)` to draw the contour of a light bulb. Here is how it should work: Make `r` steps forward, then go to coordinates `(2*r, r)`. Then move by `3*r` up. Add a right 50-degree arc of radius `6*r`. Last, add a left 145-degree arc of radius `6*r`. The contour for sample value `r = 10` is shown below:"

Number of lines in the function: 10

Function commands: `go()`, `goto()`, `angle()`, `arc()`, `color()`



The lightbulb function makes use of different commands and algebraic relationships to locate the starting point and describe distances and radii. Why is `angle()` used instead of specifying the number of degrees of turn? (The degrees may vary depending on the value of the r) Why is `goto()` used? (Again, the starting point for the next line may vary depending on the value of r). Students can experiment with the value of r to see how this works. The lightbulb will maintain its proportions and attributes regardless of the size.

9.6 Yo-Yo

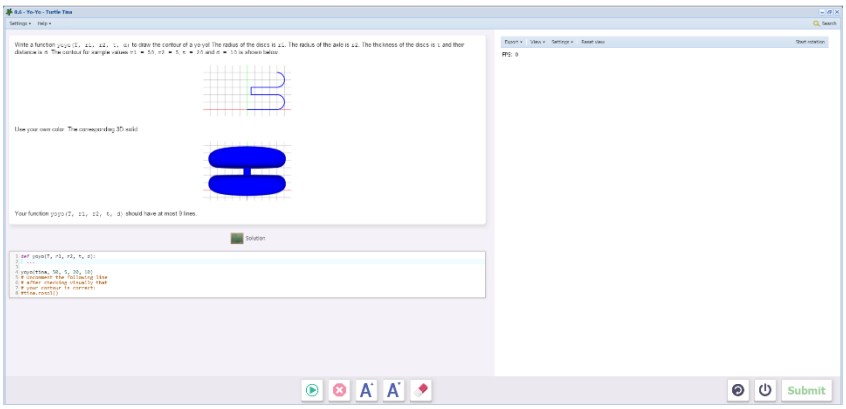
Students practice writing a function to use with `rosol()` by creating a yo-yo. Here again are the screen instructions.

"Write a function `yoyo(T, r1, r2, t, d)` to draw the contour of a yo-yo! The radius of the discs is `r1`. The radius of the axle is `r2`. The thickness of the discs is `t` and their distance is `d`. The contour for sample values `r1 = 50, r2 = 5, t = 20` and `d = 10` is shown below:

A key to writing the distances and arc radii is using ½ the thickness `(t)` as a measurement.

Number of lines in the function: 9

Commands: `go()`, `right()`, `arc()`



9.7 Tower of Hanoi

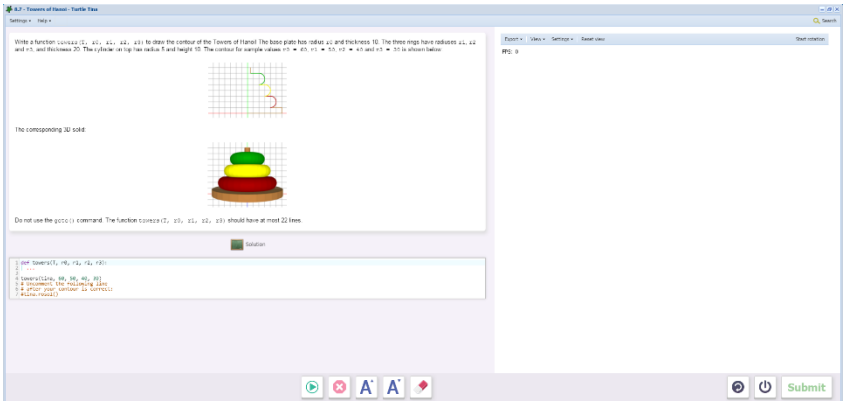
Students practice writing a function for the `rosol()` command by drawing a children's toy known as the Tower of Hanoi. Here are the screen instructions:

"Write a function `towers(T, r0, r1, r2, r3)` to draw the contour of the Towers of Hanoi! The base plate has radius `r0` and thickness 10. The three rings have radiuses `r1, r2` and `r3`, and thickness 20. The cylinder on top has radius 5 and height 10. The contour for sample values `r0 = 60, r1 = 50, r2 = 40` and `r3 = 30` is shown below:"

The code for the rings is similar to yo-yo.

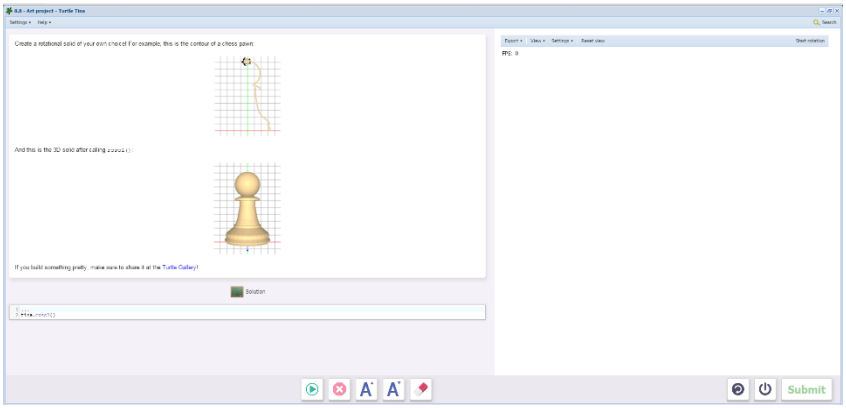
Number of lines in the function: 22

Commands: `go()`, `right()`, `arc()`



9.8 Art Project

Students create a 3D shape based on a trace built along the y-axis. The `rosol()` command will convert the trace to a printable 3D shape. Students are responsible for writing the entire code this time Encourage students to visit the Turtle Gallery and submit their own drawings for publication.

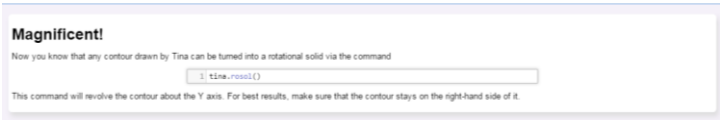


The program should include a function with variables, and the `rosol()` command.

Students should sketch ideas on graphs and make notes of lengths, angles, and curves. This will help plan the code.

Using variables: How are the elements of the design related? Are there common lengths or angles? Are there lengths or angles that are proportional to others? Are angles complementary or supplementary? Writing the code with variables can preserve these relationships in the function, while allowing for different input values in the main program.

After submitting the design, students will see a review of the new ideas they have learned on the next screen.



They will also earn a purple belt of third degree, which can be printed, saved, emailed or shared.

Focus Questions for post-session discussion:

Review the steps needed to create a 3D solid. (Build a trace along the y-axis; use `rosol()` to turn it into a solid.)

What are some limitations to using `rosol()`? (The shape must be symmetrical. It only rotates around the y axis in Tina 2)

What could you model using 3D solid shapes?

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. Students will receive a printable "Purple Belt of Third Degree" certificate upon completion of Section 8. See Assessment section for journal and project ideas.

As in other sections, journals and art projects can be used as assessments. The Assignment card follows:

END OF SECTION 9: ART PROJECT (50 POINTS)

Objective: Create and publish a design to make a 3D rotational solid in Tina Turtle. You will need to include a function for the trace, and the command `rosol()` to turn the trace into a solid.

Planning (15 points): After you have decided on a design, draw out the shapes on grid paper or in your journal.

- Describe the design. What commands will be needed? Name the object or pattern that you are designing.
- Looking at the design, decide which commands will be used. Will you need the `back()` command to start the next part of the pattern? Where will you use `goto()` instead of `go()`?
- If your design is composed of more than one repeated shape or pattern, write out the order in which you will place them in the program.

Writing the code (25 points): Write the program.

- Remember to include comment lines that describe what your program is doing.
- Write the code and test as you go. Test the trace before running the program with `rosol()`.
- Use variables whenever possible in your design. How are the elements of the design related? Are there common lengths or angles? Are there lengths or angles that are proportional to others? Are angles complementary or supplementary? Writing the code with variables can preserve these relationships in the function, while allowing for different input values in the main program.

- Use some or all of the commands learned so far:
  `go(), left(), right(), back(), pu() or penup(), pd() or pendown(), goto(), home(),angle(), arc(), color(), rosol()`

Saving the file, sharing and publishing (10 points)

- Publish the design to your folder. Inform someone else about the design by providing the link on _____, or by email.

SECTION 10 SURFACES AND SHELLS: LEVELS 10.1-10.8

Objectives: In this section, students learn to create rotational surfaces and shells.

Vocabulary:

Rotational surface: a surface created by rotating a trace around an axis. The trace can be composed of line segments, arcs or combinations of these.

rosurf: a command written as tina.`rosurf`(), which rotates a line trace about the y-axis to create a 3-dimensional surface. It is written after the commands used to create the line trace. Think of the distance of the line from the y-axis as a radius.

Rotational shell: a shell created by rotating a trace around an axis. The trace can be composed of line segments, arcs or combinations of these.

roshell: a command written as tina.`roshell`(), which rotates a line trace about the y-axis to create a 3-dimensional surface. It is written after the commands used to create the line trace. Think of the distance of the line from the y-axis as a radius, similar to `rosol` and `rosurf`. **The advantage of** `roshell` **over** `rosurf` **is that** `roshell` **can be extruded and printed. The disadvantage is that it takes longer to compute.**

Time required: Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours, with another hour for the Art Project.

Prerequisite skills:

Completion of Section 9. Review geometry and algebra concepts as needed.

Background knowledge/Introductory Set/Purpose:

Shapes are often hollow. In nature, think of reeds, bamboo, shells, or even guts. Examples of crafted items include jars, bottles, bowls, vases, and other containers. Pipes and tubes are hollow. Shapes can be cones or more complex shapes: think of speaker cones and lampshades.

In programming, we need two types of hollow shape.

One is just a surface, which is easy to compute. Surfaces are useful for illustrations and models that will not be printed.

The other is a shell, which has a thickness. This takes more computing resources to build, but can be printed. In Tina, we use the extrude command to specify the thickness.

Purpose: Section 10 (Levels 10.1-10.8) Learn the commands to create surfaces and shells.

Direct Instruction and Modeling

Show the video in 10.1 that explains how to create solids and surfaces.

https://youtu.be/NrTbw57jTUE

The first level can be modeled to the class, using a smartboard or projector.

The help card for Section 10 is on the next page.

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Creating Rotational Surfaces and Shells using `rosurf` and `roshell` (Tina 2, Section 10)

Rosurf command: The command is written as `rosurf()`

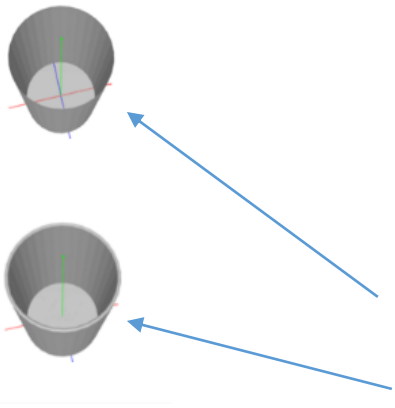
Roshell command: The command is written as `roshell()`

Both commands create a hollow, rotational 3D shape. Rosurf creates a thin surface, which is fast to compute and useful for drawings or computer models. Roshell is thicker and can be printed, using the `extrude`() command to give a value to the thickness.

The shapes are created in the same way as Rosol.

Create a set of commands that will draw a line trace. Follow this set with the `rosurf()` or `roshell()` command to rotate the shape around the y axis.

Example:



```
tina.color(STEEL)
tina.go(30)
tina.left(90)
tina.go(80)
tina.go(30)

tina.rosurf()
or
tina.rosol()
```

.

Self-paced Instruction: Levels 10.1-10.8 (All previous commands may be needed but not necessarily listed under each lesson. Defined objects are listed)

10.1 Water Glass

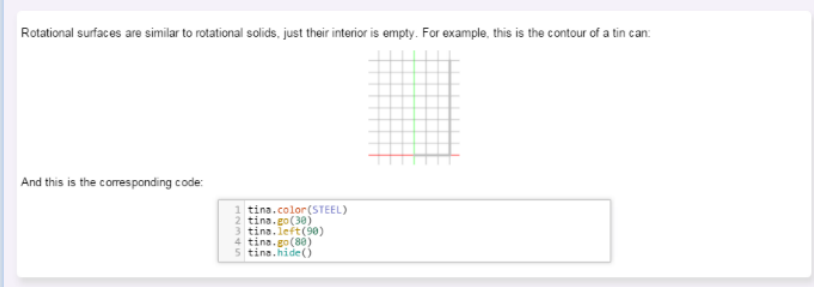
10.1 starts with a YouTube video on nested loops, which can be watched from within the program or by following this link:

https://youtu.be/NrTbw57jTUE

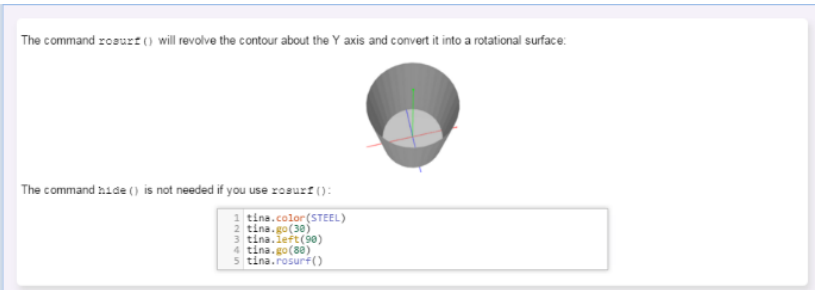
This video demonstrates how to build rotational surfaces and shells. The process is similar to building a rotational solid. First, a trace is built along the y-axis. Then, it is converted to a rotational surface using the command `rosurf()`. This command draws quickly, but the resulting surface is too thin to print on a 3D printer. If a printable object is desired, then the width must be specified and the command rotate() used.

The next three screens demonstrate how to build a tin can (a hollow cylinder).

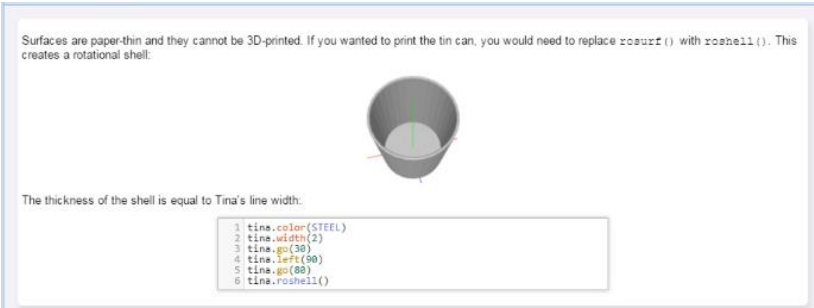
First, the trace is drawn.



Then, the command `rosurf()` is used to convert the trace to a rotational surface.



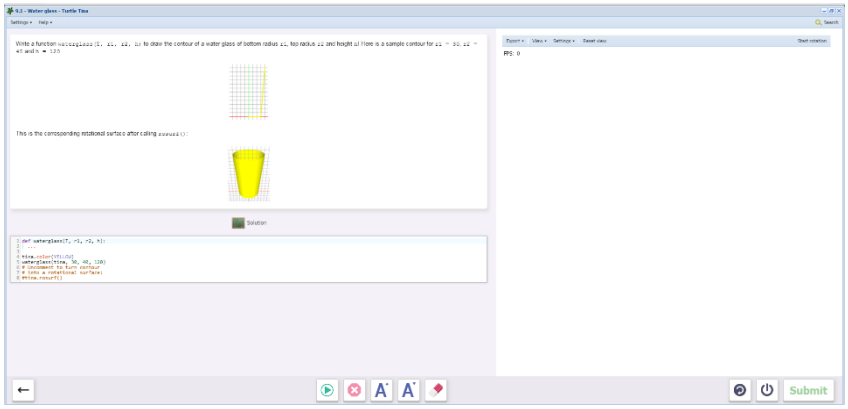
If a printable solid is desired, then the `roshell()` command is used instead. The width must be specified.



To practice this skill, students build a water glass by writing a function

`waterglass(T, r1, r2, h)`, where `r1` is the radius of the base, `r2` is the radius of the top, and `h` is the height,

and calling it in the main program using `rosurf()`



Number of lines in the function: 2

Commands: `go(), goto()`

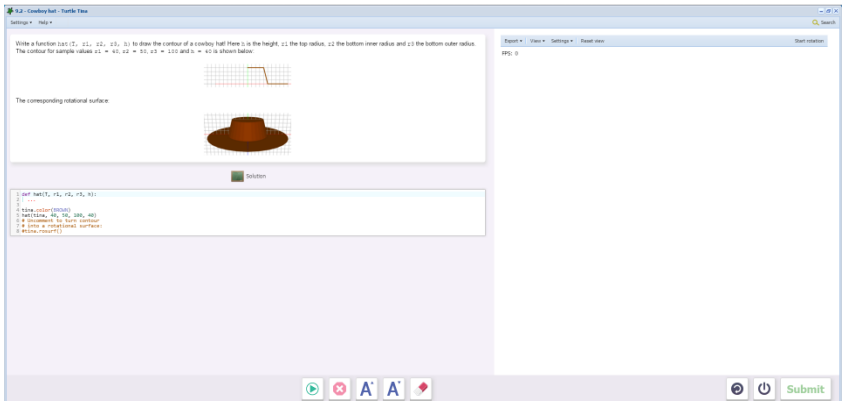
10.2 Cowboy Hat

Students create a hollow hat using a function and the `rosurf()` command.

This time, 3 radii are needed in the function.

Hint: start drawing at the top of the hat for a simpler script.

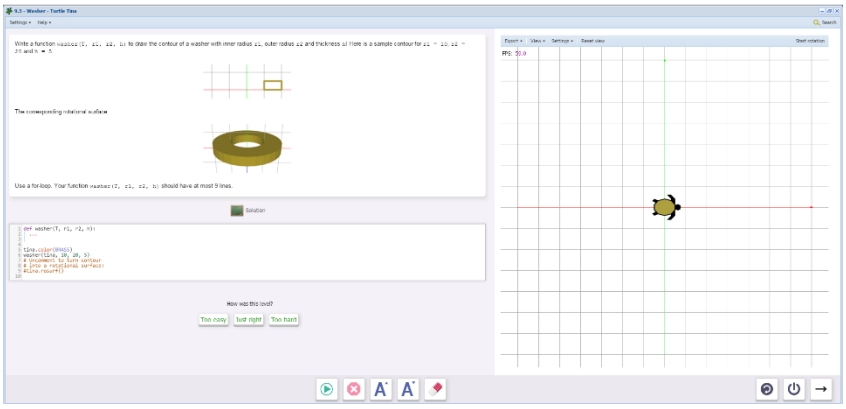
Number of lines in the function: not specified. The suggested solution has 9 lines.



Commands: `go(), left(), right(), goto(), pu() or penup(),pd() or pendown(), angle()`.

10.3 Washer

Students practice using `rosurf()` by creating a function to draw a washer. A simple rectangle turns into a washer!



Use a For loop to create the rectangle.

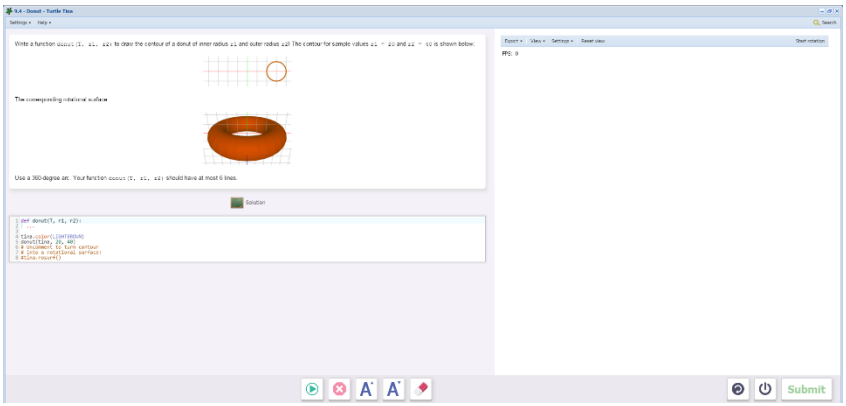
The dimensions are r1 (inner radius), r2 (outer radius), and height.

Number of lines in the function: 9

Commands: `go()`, `left()`, `pu()` or `penup()`, `pd()` or `pendown()`

10.4 Donut

Students practice `rosurf()` by creating a function to draw a donut. The cross-section of the donut is a 360 degree arc.



The inner radius, r1, and outer radius, r2, are specified.

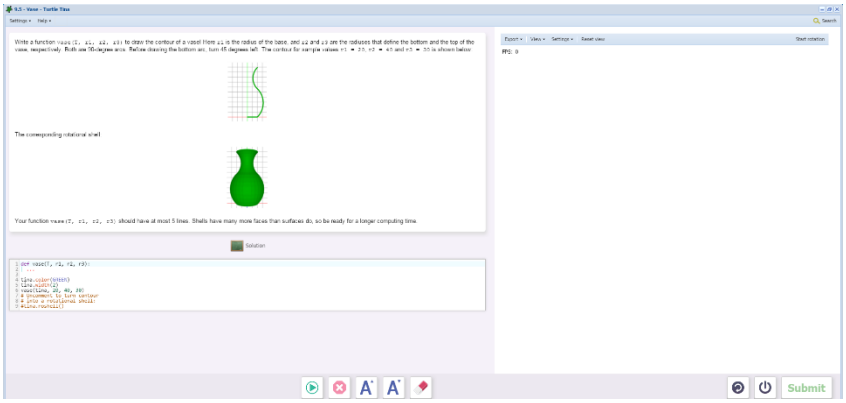
Number of lines: 6

Commands: `go()`, `left()`

10.5 Vase

Students practice `roshell()` by creating a function to draw a vase.

Here are the screen directions: “Write a function `vase(T, r1, r2, r3)` to draw the contour of a vase! Here `r1` is the radius of the base, and `r2` and `r3` are the radiuses that define the bottom and the top of the vase, respectively. Both are 90-degree arcs. Before drawing the bottom arc, turn 45 degrees left. The contour for sample values `r1 = 20`, `r2 = 40` and `r3 = 30` is shown below:”



Number of lines: 5

Commands: `go()`, `left()`, `right()`

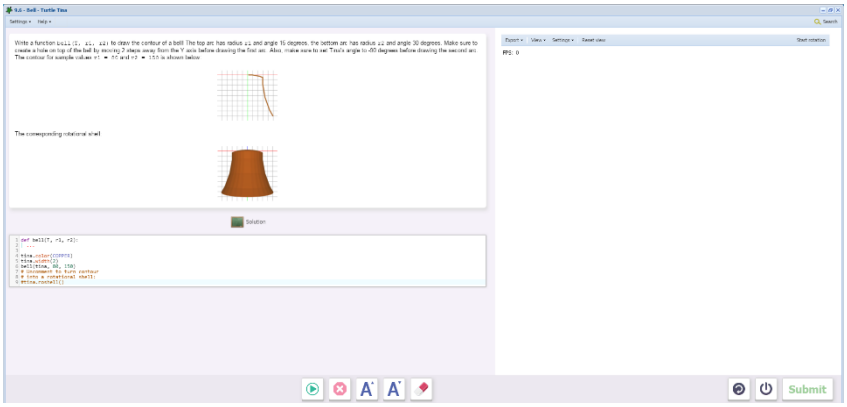
Shells take longer to compute, so be prepared for long wait times.

Ask students if they can see the 90 arcs, even though only portions of the arcs are actually drawn.

10.6 Bell

Students practice `roshell()` by creating a function to draw a bell.

Here are the screen directions: “Write a function `bell(T, r1, r2)` to draw the contour of a bell! The top arc has radius `r1` and angle 15 degrees, the bottom arc has radius `r2` and angle 30 degrees. Make sure to create a hole on top of the bell by moving 2 steps away from the Y axis before drawing the first arc. Also, make sure to set Tina's angle to -90 degrees before drawing the second arc. The contour for sample values `r1 = 80` and `r2 = 150` is shown below:”



Note that the bell trace is drawn in the 2nd quadrant (below the x-axis).

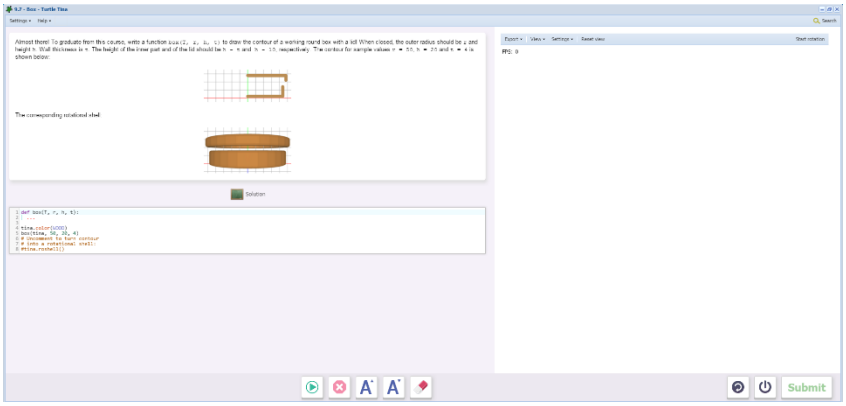
Number of lines in the function: 8

Commands: `go()`, `pu()` or `penup()`, `pd()` or `pendown()`, `angle()`, `arc()`

10.7 Box

Students practice `roshell()` by creating a function to draw a box with a lid.

Here are the screen directions: "…Write a function `box(T, r, h, t)` to draw the contour of a working round box with a lid! When closed, the outer radius should be `r` and height `h`. Wall thickness is `t`. The height of the inner part and of the lid should be `h - t` and `h - 10`, respectively. The contour for sample values `r = 50`, `h = 20` and `t = 4` is shown below:



Drawing the two shapes looks simple, but precision is involved in order for the lid to fit snugly on the box. The line lengths and heights must take the thickness t into consideration.

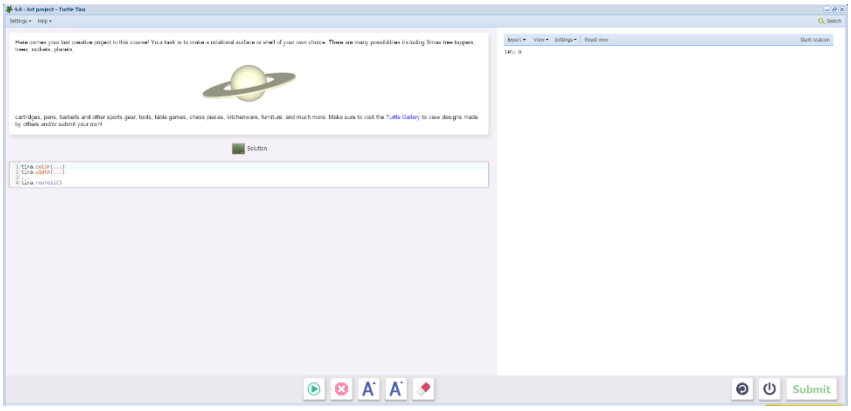
A lidded, cardboard or wooden box from a craft show will be useful to demonstrate this precision.

Number of lines in the function: 19

Commands: `go(), left(), right(), pu() or penup(), pd() or pendown()`

10.8 Art Project

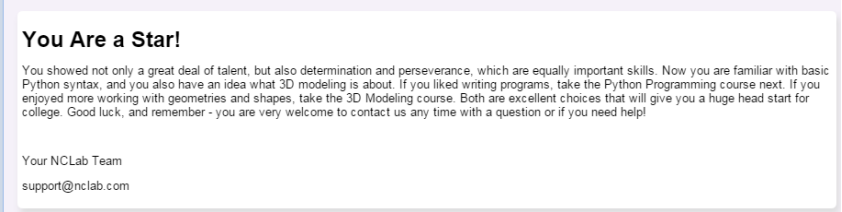
Students create their own rotational surface or shell.



Reminder: patterns must be extruded in order to print as a 3D object.

Files should be saved, then shared. Files can be submitted to the Tina Turtle Gallery.

Upon successful submission of the Art Project, students will see this screen.



Students will also receive a Purple Belt of the 4th degree, which they can print, save, or share.

Focus questions for post-session discussion:

What are the differences between `rosurf()` and `roshell()`? When would you use one or the other? (`rosurf()` is faster and works for designs that won't be printed. `roshell()` is needed to make objects thick enough to print. You can also control the thickness using the `extrude()` command.)

What applications might use rotational surfaces and shells? (3D printing, animation, engineering design)

Go back and experiment with the variables. How does this affect the results? This could be a team project with each person on the team demonstrating a different set of variables. Then, the whole class could walk around and look at each other's results.

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. Students will receive a printable "Purple Belt of Fourth Degree" certificate upon completion of Section 10. See Assessment section for journal and project ideas.

Students Journal entries can be used as assessment.

The Art Project can be used as a performance task or portfolio artifact. On the next page is a possible 50-point assignment card.

What's Next?

Students who successfully complete Tina 2 will be ready to study 3D design and learn the full Python Language. NCLabs offers self-paced courses in both: 3D Design 1, 2, and 3, and Python 1 and 2.

Students are encouraged to submit designs to the NCLab Gallery. This is a chance to refine and display their design skills.

END OF SECTION 10: ART PROJECT (50 POINTS)

Objective: Create and publish a design to make a 3D rotational surface or shell in Tina Turtle. You will need to include a function for the trace, and the command `rosurf()` to turn the trace into a surface, or `roshell()` to turn the trace into a shell.

Planning (15 points): After you have decided on a design, draw out the shapes on grid paper or in your journal.

- Describe the design. Remember that the trace only shows edges and cross-sections.
- Name the function.
- Look for proportions and relationships between different lengths and between different angles. How will you make use of variables in your code?
- In what order and direction will you draw the components? How will you go from one component to another? When will you use commands such as `pu() or penup(), pd() or pendown(), goto(), back(), home()`?

Writing the code (25 points): Write the program.

- Remember to include comment lines that describe what your program is doing.
- Write the code and test as you go.
- Use at least one function and either `rosurf()` or `roshell()` in the main program.
- Use variables whenever possible in your design. Writing the code with variables can preserve these relationships in the function, while allowing for different input values in the main program.

- Use some or all of the commands learned so far:
`go(), left(), right(), back(), pu() or penup(), pd() or pendown(),goto(), home(),angle(), arc(), color()`

Saving the file, sharing and publishing (10 points)

- Publish the design to your folder. Inform someone else about the game by providing the link on _____, or by email.

NOTES