

TEALS Program

[Home](#) | [Curriculum Map](#)

1 About the AP Computer Science A Curriculum

The [TEALS Program](#) has designed these curriculum materials for the use of teachers and volunteer tech professionals in high school classrooms. Any teacher with prior programming experience (or access to a computer science professional) can use this curriculum to teach the AP Computer Science A course.

This curriculum is based on and aligned with Professor Stuart Reges' course at the University of Washington, CSE 142. The course uses the textbook:

Building Java Programs: A Back to Basics Approach by Stuart Reges and Marty Stepp. Publisher: Pearson; 5 edition (March 28, 2019) ISBN-10: 013547194X ISBN-13: 978-0135471944

The course is aligned with the AP Computer Science A standards. TEALS has received AP Audit certification for previous versions of the course and syllabus.

The TEALS AP curriculum was approved by the CollegeBoard so partner schools may use the "claim identical" function of the AP Audit website to obviate the need for their own curriculum audit. Specific instructions are available in the [AP CS A Course Audit Instructions](#).

This curriculum uses principles of universal design for learning (UDL). The curriculum was written for and tested in classrooms with diverse learners; students with individualized education plans, English language learners, students who have received sub-optimal math or language instruction in the past, students who are gifted/talented, students who are otherwise "outside the average." See [Additional Resources](#) for more information on universal design for learning.

1.1 Accessing the Curriculum

[College Board endorsed curriculum sequence \(2020\)](#) This curriculum map follows the College Board Sequence of lessons.

[Textbook sequence \(before 2020 school year\)](#) This curriculum map follows the Building Java Program Textbook sequence of lessons

The AP Computer Science A Curriculum is located on GitHub pages:

<https://tealsk12.github.io/apcsa-public/>

The AP Computer Science A Curriculum GitBook has been deprecated:

<https://www.gitbook.com/book/tealsk12/ap-computer-science-a/details>.

For contributions to the curriculum, the AP Computer Science A GitHub repository is located at

<https://github.com/TEALSK12/apcsa>.

1.2 Using the curriculum

Each classroom has different physical, cultural, academic, and scheduling needs. Therefore, we have tried to create a collection of lessons and materials that are adaptable to most situations. TEALS volunteers and

classroom teachers will find different aspects of the curriculum useful; you should expect to skip over certain notes to focus on the information that is most useful to you.

We have provided classroom management tips and engagement tips for TEALS volunteers, who are new to the classroom setting. Experienced teachers and volunteers will likely choose to skip such details and focus on the step-by-step lecture notes.

You may browse the [Curriculum Map](#) for an overview of the pacing, objectives, and assessments.

1.2.1 Year Round Pacing

The table-of-contents (included with Introduction materials) contains coarse-grained time estimates on the scale of weeks and days so teachers can plan accordingly. Units 6 and 8 include extra days in the time-estimate so teachers can re-adjust their unit plans if they have shifted due to unexpected class cancellations or drift.

1.2.2 Daily Structure

Every classroom is different, and we expect that instructors will adapt the daily structure of the class to suit their students' needs. That said, we've designed most of the lessons using the following daily structure:

1.2.2.1 Hook & Instruction

- Each lesson plan begins with one or several options for short (from seconds to 5 minutes) engaging or mystifying activities that introduce students to the topics to be introduced later in the lesson.
- Lecture notes, student prompts, and quick-assessments (with answers) are outlined in subsection "Introduction." If you are teaching in a flipped classroom, this section can be pre-recorded for students to view at home. For additional resources on flipping your classroom, please refer to "Additional Resources" below.

1.2.2.2 Student Practice

- Student practice/activities are outlined with step-by-step instructions including pacing suggestions and alternative stopping points. Any special materials or preparation needed for the hook, lecture, or activity are listed in the Materials & Prep section.

1.2.2.3 Warmup / DoNow / Boardwork/Ticket-to-leave

- Since each classroom progresses at different rates, we have not included warm-up and cool-down questions (though time has been scheduled in the Pacing Guide for one or both of these activities). You should choose your questions based on the topics you felt were most challenging or confusing for your students. A good source for short-answer and multiple choice questions is the [Barron's AP Computer Science A review book](#), which TEALS ships to each AP CS A volunteer.

1.2.3 Scaffolding

The Glossary of Education Reform defines scaffolding as:

A variety of instructional techniques used to move students progressively toward stronger understanding and, ultimately, greater independence in the learning process.

Instructors provide successive levels of temporary support that help students reach higher levels of comprehension than they would have been able to achieve without assistance. Support is gradually removed as

students move towards mastery, which occurs when students demonstrate skills and knowledge without any outside assistance.

The University of Washington course CSE 142 and associated textbook do not contain much scaffolding. This curriculum attempts to wrap the content of the UW course with scaffolding appropriate for high school classes. Some classes may not require scaffolding, and other classes may need even more scaffolding than those steps suggested within the lesson plan.

1.2.4 Examples

Most lecture notes and classroom examples are slightly modified versions of the examples outlined in the textbook. When the class needs additional examples, or re-teaching, instructors can refer directly to the textbook for a fresh set of similar examples and explanations. The “additional resources” section of this document lists some other sources for examples and labs.

1.2.5 References to the textbook

Some classrooms are using earlier editions of the Building Java Programs textbook. To avoid confusion, we have written all reading and practice assignments by chapter and section rather than page number. In cases where practice problems or assignments differ between editions, we have copied those assignments (with reference) into printable documents.

1.2.6 Homework Assignments

As written, the homework assignments contain material to be assigned, but are not phrased in terms of learning goals. Teachers should choose specific learning goals for the evening's work depending on student progress and timing within the week and school year, then phrase the assignment in terms of learning goals, not output.

For example, rather than “read section 3.1” assign the reading by saying “for tomorrow, be prepared to pass data into methods using parameters. Section 3.1 in the textbook will show you how.”

1.2.7 Pokémon

Throughout the course, this curriculum includes lab assignments using the Pokémon universe as a subject-matter domain (often replacing textbook assignments on less salient topics like compound interest). The Pokémon storyline and game rules are familiar to male and female students from all socioeconomic backgrounds, available across the digital divide as both a card game and a video game, and are available in 10 different languages (English, Spanish, Portuguese, Dutch, French, German, Italian, Korean, Chinese, and Japanese).

Because the game relies on statistics, modulo operators, and the underlying 32-bit integer that characterizes any given Pokémon, we will be using this theme to introduce students to much of the AP CS A curriculum. Students will be entering the AP CS A course with varying degrees of math literacy, and framing mathematical challenges in this familiar framework is helpful for avoiding stereotype threat and math anxiety.

To learn more about the Pokémon storyline, game rules, underlying formulae, and characters, visit:

<http://bulbapedia.bulbagarden.net/>. For a more general introduction to the Pokémon franchise, visit:

<http://www.pokemon.com/>.

1.2.8 AP Test Preparation

Aligned to the College Board's curriculum framework, students explore the big ideas that encompass the core principles, theories, and processes of computer science. Throughout the course, the student learns and practices the skills necessary to be successful on the AP exam.

1.2.8.1 Big Ideas of Computer Science

1. Modularity — Incorporating elements of abstraction, by breaking problems down into interacting pieces, each with their own purpose, makes writing complex programs easier. Abstracting simplifies concepts and processes by looking at the big picture rather than being overwhelmed by the details. Modularity in object-oriented programming allows us to use abstraction to break complex programs down into individual classes and methods.
2. Variables — Information used as a basis for reasoning, discussion, or calculation is referred to as data. Programs rely on variables to store data, on data structures to organize multiple values when program complexity increases, and on algorithms to sort, access, and manipulate this data. Variables create data abstractions, as they can represent a set of possible values or a group of related values.
3. Control — Doing things in order, making decisions, and doing the same process multiple times are represented in code by using control structures and specifying the order in which instructions are executed. Programmers need to think algorithmically in order to define and interpret processes that are used in a program.
4. Impact of Computing — Computers and computing have revolutionized our lives. To use computing safely and responsibly, we need to be aware of privacy, security, and ethical issues. As programmers, we need to understand how our programs will be used and be responsible for the consequences.

1.2.8.2 Computational Thinking Practices: Skills

1. Program Design and Algorithm Development A. Determine an appropriate program design to solve a problem or accomplish a task (not assessed by AP Exam). B. Determine code that would be used to complete code segments. C. Determine code that would be used to interact with completed program code.

Curriculum Example of Skill 1.B: In Lesson 1.06, students are challenged to design and write a class that reproduces a particular shape pattern that encourages decomposition into multiple static methods.

2. Code Logic A. Apply the meaning of specific operators. B. Determine the result or output based on statement execution order in a code segment without method calls (other than output). C. Determine the result or output based on the statement execution order in a code segment containing method calls. D. Determine the number of times a code segment will execute.

Curriculum Example of Skill 2.A: In Lesson 2.01, students are introduced to the modulus operator and practice evaluating expressions that use it.

3. Code Implementation A. Write program code to create objects of a class and call methods. B. Write program code to define a new type by creating a class. C. Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements. D. Write program code to create, traverse, and manipulate elements in 1D array or ArrayList objects. E. Write program code to create, traverse, and manipulate elements in 2D array objects.

Curriculum Example of Skill 3.C: In Lesson 2.08, students write a method to reproduce an hourglass shape using nested for loops.

4. Code Testing A. Use test-cases to find errors or validate results. B. Identify errors in program code. C. Determine if two or more code segments yield equivalent results.

Curriculum Example of Skill 4.B: In Lesson 3.09, students are asked to correct syntax errors in a series of conditional statements.

5. Documentation A. Describe the behavior of a given segment of program code. B. Explain why a code segment will not compile or work as intended. C. Explain how the result of program code changes, given

a change to the initial code. D. Describe the initial conditions that must be met for a program segment to work as intended or described.

Curriculum Example of Skill 5.A: In Lesson 3.12, students are tasked with starting a program that will be finished by partner, with the only communication between them being well-commented code.

1.2.8.3 AP Exam Preparation

All of the Unit tests are in the AP exam format. In classes where many students will take the exam, instructors should gradually adjust the testing environment to mimic that of the exam:

- Always provide/allow the [AP Java Quick Reference](#)
- Move from open-note (see “Tricky Code Cheat Sheet”) to closed-note
- The AP exam has 40 multiple choice questions in 90 minutes (\approx 2 minutes per question). On the earlier tests, start at a slower pace (perhaps 4 minutes per question). As the course progresses, work to a pace even faster than the actual test (90 seconds per question).

1.2.9 Vocabulary

A comprehensive vocabulary list for each unit is provided for teachers to generate word walls in their classroom. Some classrooms will be able to omit certain vocabulary words; as offered, the lists offered include words that English language learners and students with previous sub-optimal instruction may find challenging.

1.2.10 Error-Checking Lessons

One class period in each unit has been devoted to student correction and resubmission of work. While it may be tempting to “win back” class time by skipping these sessions, we strongly encourage teachers to leave these sessions in.

When students have the opportunity to fix their work and earn back full or partial credit, it gives students agency over their grade and teaches students to examine and reflect upon their own learning. On a practical note, when error-checking lessons are included, teachers need only grade answers as correct/incorrect, since students will be challenged with finding and fixing the errors on their own later. Finally, students that have answered all or most of their work correctly receive a day off to do silent work/play on their own, which positively reinforces students to put in the initial effort to win a day off.

1.2.11 AP Test Preparation

Aligned to the College Board's curriculum framework, students explore the big ideas that encompass the core principles, theories, and processes of computer science. Throughout the course, the student learns and practices the skills necessary to be successful on the AP exam.

1.2.11.1 Big Ideas of Computer Science

1. Modularity — Incorporating elements of abstraction, by breaking problems down into interacting pieces, each with their own purpose, makes writing complex programs easier. Abstracting simplifies concepts and processes by looking at the big picture rather than being overwhelmed by the details. Modularity in object-oriented programming allows us to use abstraction to break complex programs down into individual classes and methods.
2. Variables — Information used as a basis for reasoning, discussion, or calculation is referred to as data. Programs rely on variables to store data, on data structures to organize multiple values when program

complexity increases, and on algorithms to sort, access, and manipulate this data. Variables create data abstractions, as they can represent a set of possible values or a group of related values.

3. Control — Doing things in order, making decisions, and doing the same process multiple times are represented in code by using control structures and specifying the order in which instructions are executed. Programmers need to think algorithmically in order to define and interpret processes that are used in a program.
4. Impact of Computing — Computers and computing have revolutionized our lives. To use computing safely and responsibly, we need to be aware of privacy, security, and ethical issues. As programmers, we need to understand how our programs will be used and be responsible for the consequences.

1.2.11.2 Computational Thinking Practices: Skills

1. Program Design and Algorithm Development A. Determine an appropriate program design to solve a problem or accomplish a task (not assessed by AP Exam). B. Determine code that would be used to complete code segments. C. Determine code that would be used to interact with completed program code.

Curriculum Example of Skill 1.B: In Lesson 1.06, students are challenged to design and write a class that reproduces a particular shape pattern that encourages decomposition into multiple static methods.

2. Code Logic A. Apply the meaning of specific operators. B. Determine the result or output based on statement execution order in a code segment without method calls (other than output). C. Determine the result or output based on the statement execution order in a code segment containing method calls. D. Determine the number of times a code segment will execute.

Curriculum Example of Skill 2.A: In Lesson 2.01, students are introduced to the modulus operator and practice evaluating expressions that use it.

3. Code Implementation A. Write program code to create objects of a class and call methods. B. Write program code to define a new type by creating a class. C. Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements. D. Write program code to create, traverse, and manipulate elements in 1D array or ArrayList objects. E. Write program code to create, traverse, and manipulate elements in 2D array objects.

Curriculum Example of Skill 3.C: In Lesson 2.08, students write a method to reproduce an hourglass shape using nested for loops.

4. Code Testing A. Use test-cases to find errors or validate results. B. Identify errors in program code. C. Determine if two or more code segments yield equivalent results.

Curriculum Example of Skill 4.B: In Lesson 3.09, students are asked to correct syntax errors in a series of conditional statements.

5. Documentation A. Describe the behavior of a given segment of program code. B. Explain why a code segment will not compile or work as intended. C. Explain how the result of program code changes, given a change to the initial code. D. Describe the initial conditions that must be met for a program segment to work as intended or described.

Curriculum Example of Skill 5.A: In Lesson 3.12, students are tasked with starting a program that will be finished by partner, with the only communication between them being well-commented code.

1.2.11.3 AP Exam Preparation

All of the Unit tests are in the AP exam format. In classes where many students will take the exam, instructors should gradually adjust the testing environment to mimic that of the exam:

- Always provide/allow the [AP Java Quick Reference]
- Move from open-note (see “Tricky Code Cheat Sheet”) to closed-note

- The AP exam has 40 multiple choice questions in 75 minutes (\approx 2 minutes per question). On the earlier tests, start at a slower pace (perhaps 4 minutes per question). As the course progresses, work to a pace even faster than the actual test (90 seconds per question).

1.3 Video Tutorials

- [Timing and Pacing](#) — Adjusting lessons and the curriculum map for the speed of your learners
- [Projects and Labs](#) — Choosing whether your class completes the AP labs or the projects (FracCalc/TextExcel)
- [Supporting Visual-Spatial Learners](#) — Using the physical space in your classroom to enhance learning
- [Parson's Problems](#) — Assessing high-level programming skills quickly with Parson's Problems
- [Grudgeball](#) — Reviewing material by playing a class game of Grudgeball

1.3.1 Recommended Hardware

In the classroom, it is recommended that each student have an internet-connected desktop computer capable of running an Integrated Design Environment (IDE). Students will need to be able to save and access their programming projects locally or in the cloud.

1.3.2 Integrated Design Environment (IDE)/Code Editor

Coding in Java requires the Java Development Kit and a text editor or IDE. There are many Java IDEs available. There are any IDEs/Coded editors that can be used to teach AP CS A. Unit 1 includes directions for installing IDE/Code Editor for Eclipse and VS Code. You are free to choose the IDE/Code Editor that best fits your class.

1.3.3 Textbook

The TEALS curriculum requires each student to have a copy of the textbook. Many assignments require students to complete self-checks, exercises, and programming problems at the end of each chapter from the textbook. While Practice-it is available from the University of Washington, it is not necessary. As with all software services, it is the school's sole decision to use the tool according to the use terms and privacy policies provided by its licensor and it is the school's responsibility to ensure the tool meets its IT policies.

1.3.4 Detecting Cheating with MOSS

Although the curriculum does not specifically outline an approach for monitoring cheating, many teachers have found it easier, faster, and less stressful to use a free plagiarism-detection program offered by Stanford at <http://theory.stanford.edu/~aiken/moss/>. Teachers will still need to manually inspect code flagged by MOSS, but the program does catch common tactics including renaming variables and reordering methods.

Occasionally, teachers have difficulty registering for an account. If this occurs, you are encouraged to email the program's creator Alex Aiken directly, at aiken@cs.stanford.edu.

1.4 Additional Resources

- The free web-based game Code Hunt (<http://www.codehunt.com>) offers opportunities for students to find and fix errors by "discovering the missing code segments." Assignments/Levels are automatically graded, and students can compete against each other to hone their programming skills.
- CodingBat (<http://www.codingbat.com>) offers Java practice problems with instant feedback for students. The problems in CodingBat are distinct from those in the Building Java Programs textbook. CodingBat

has a teacher dashboard, and a system of badges to motivate learners. Instructors can also upload their own sets of java problems for their classes to complete.

- If you are interested in learning more about principles of universal design for learning, please visit <http://www.udlcenter.org/aboutudl/udlguidelines>.
- Emerging EdTech has collected a sample of 20 digital tools to increase collaboration in the classroom. One of them might be perfect for your classroom:
- See [20 Fun Free Tools for Interactive Classroom Collaboration](#). Other tools for collaboration that have been successfully used in TEALS classrooms include Twiddla, Vyew, Skype, and Google Hangouts.
- If your classroom does not already have a digital grade management system, previous TEALS teaching teams have used Moodle, Canvas, Schoology, Excel Online, and Google Forms.
- To create digital, self-grading, and responsive quizzes, Google Forms and Socrative offer free tools and tutorials to use their systems.
- If you are stationed in a high-performing school, or in a school where many students have already mastered other programming languages, you may want to consider flipping (or inverting) your classroom. To learn more about the theory and practice of teaching in a flipped classroom, Vanderbilt University offers a comprehensive introduction and links to practical resources/examples here: <http://cft.vanderbilt.edu/guides-sub-pages/flipping-the-classroom>.

You should still be able to use most of the resources offered in this curriculum, but you will have to shuffle how you use the lesson plans. Some quick recommendations:

1. Use the lecture notes as given, but record the lecture for student viewing.
 2. Where lecture activities have been suggested (e.g. think-pair-shares), consider embedding questions into your lesson plans.
 3. Save class competitions for in-class, and leave reading and self check, and worksheet exercises for home review.
- As you read through the lesson plans, you will find several classroom teaching activities and strategies appear repeatedly. Brief video tutorials modeling these activities can be found within the TEALS repository. Keep an eye out for specific adjustments to the lesson plans for error-checking and test review. While these lesson plans look identical at first glance, small adjustments have been made for content, timing, and AP test prep.

1.5 Printing GitBook

The AP CS A GitBook can be printed by navigating to <https://aka.ms/TEALSAAPCSAPDF>. However, the "Download" button does not work. There is workaround depending on the browser:

- click on the document to enable the pdf menu to show and clicking the down arrow or "Save as Copy"
- right click on the .pdf document and select "Save As"

1.6 Giving feedback on the curriculum

TEALS intends for this curriculum to be a starting point; it's our first attempt at a complete AP CS A curriculum. We'll continue evolving and adapting the curriculum and associated materials as we learn more about teaching AP CS A. To participate in this process, we invite TEALS team members and independent teachers using this curriculum to submit edits and suggestions via the discussion forum on the TEALS dashboard or in this [Github repository issues page](#).

TEALS Program

[Home](#) | [Curriculum Map](#)

1 AP CS A Video Tutorials

The following are a set of video tutorials to help guide new teachers on the TEALS curriculum.

1.1 Videos

- Timing and Pacing: Adjust lessons and the curriculum map for the speed of your learners
- Projects and Labs: Choose whether your class completes the AP Labs or the Projects(FracCalc/TextExcel)
- Space: Using the physical space in your classroom to enhance learning
- Parson's Problems: Assessing high-level programming skills quickly with Parson's Problems
- Grudgeball: Review materials using by playing a game of Grudgeball

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 TEALS AP CS A Curriculum Assets

The TEALS AP CS A Curriculum assets may be downloaded from the [Additional Curriculum Materials](#) section of the TEALS Dashboard

[Note: you need to be a current TEALS volunteer or classroom teacher to access the TEALS Dashboard]

The latest version is TEALS-APCSA-Curriculum-v2.0.2.zip.

1.1 Contents

- /Projects/
 - /Projects/APCSA-Elevens-v1.X.X.zip The Unit 7 Elevens Lab. Extracting this archive, you'll find the Teacher Guide, and the student-distributable package. Get started by reading TeacherMaterials/Elevens-Teacher-Guide.pdf.
 - /Projects/APCSA-FracCalc-v1.X.X.zip The Unit 3 Fractional Calculator Project. Inside this archive you'll find documentation and student starter source code.
 - /Projects/APCSA-Magpie-v1.X.X.zip The Unit 4 Magpie Chatbot Lab. Inside this archive you'll find the teacher guide, teacher solution source code, and the distributable starter code archive for students. Get started by reading Magpie-Teacher-Guide.pdf.
 - /Projects/APCSA-PictureLab-v1.X.X.zip The Unit 5 Picture Lab. This archive contains the teacher guide, teacher solution code, and distributable starter package for students. Get started by reading TeacherMaterials/pixLab-Teacher-Guide.pdf.
 - /Projects/APCSA-TextExcel-v1.X.X.zip The Unit 6 Text Excel Project. This archive contains the teacher guide, teacher solution code, and distributable starter project for students. Get started by reading guides/Text Excel Teacher Guide.docx.
- /Unit*/ Assets for each of the AP CSA curriculum units. In general, each Word file will have a corresponding PDF equivalent. Worksheets are generally of the form "WS #.#.docx" and "WS #.#.pdf".
 - /Unit1/ — Assets for Unit 1: Programming & Java.
 - /Unit2/ — Assets for Unit 2: Working with Data & Basic Control Flow.
 - /Unit3/ — Assets for Unit 3: Advanced Data & Control Flow.
 - /Unit4/ — Assets for Unit 4: Arrays, Lists & Files.
 - /Unit5/ — Assets for Unit 5: Object-Oriented Programming.
 - /Unit6/ — Assets for Unit 6: Inheritance & Polymorphism.
 - /Unit7/ — Assets for Unit 7: Searching & Sorting.
 - /Unit8/ — Assets for Unit 8: Recursion.
 - /Unit9/ — Assets for Unit 9: AP Test Review.

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Unit 1: Programming & Java (2 weeks)

The following curriculum map is a day-by-day listing of the AP Computer Science course in chronological order. Each row represents one day of class, based on a medium-paced class. Readings from the textbook and homework assignments are included on the day when they should be assigned. Refer to the Introduction document for information about how to adjust this pacing for your specific classroom.

- [Unit 1 Slides](#)
- [Unit 1 Word Bank](#)
- [Curriculum Assets](#)

LP	Title	In Class	Reading	Homework
1.01	<i>Using IDE/Code Editor</i>	WS 1.1.1 WS 1.1.2		Explore PokéMón (pokemon.com, console, web site)
1.02	<i>Algorithms & Computational Thinking</i>	PB&J		Reflect on PB&J assignment Continue exploring PokéMón
1.03	<i>String & Console Output</i>	SC 1.6–9, 1.11–14 LP Welcome	1.2	EX 1.1–5
1.04	<i>Common Errors & Comments</i>	WS 1.4	1.3	EX 1.6–9
1.05	<i>Static Methods & Method Calls</i>	SC 1.22, 1.23, 1.26, 1.29	1.4	EX 1.11, 1.12, 1.14, 1.16
1.06	<i>Static Methods & Method Calls</i>	LP StarFigures LP PikachuChatter	1.5	Outline ch 1 PP 1.1, 1.3
1.07	<i>Programming Project</i>	PP 1.2, 1.5		Note check (add summaries if needed)
1.08	<i>Finding & Fixing Errors</i>	Fix homework		Submit questions for review
1.09	<i>Review</i>	Review questions WS 1.9 Practice test	Review ch 1	Study
1.99	<i>Unit 1 Test</i>	Test 0 Section I Test 0 Section II		
1.XX	<i>Lesson 1.07 Alternative</i>	Ideate and Construct Project		

1.1 1.01

Lesson 1.01	<i>Using IDE/Code Editor</i>
Objectives	Students will be able to open IDE/Code Editor, create and save a file in IDE/Code.
Assessments	Students will demonstrate Plug-In and Un-Plug procedures for the teacher. Students will log in and submit a sample problem in Practice.
In Class	WS 1.1.1 WS 1.1.2
Reading	
Homework	Explore Pokémon (pokemon.com , console, web site)

1.2 1.02

Lesson 1.02	<i>Algorithms & Computational Thinking</i>
Objectives	Students will be able to define algorithms, programs, hardware, software, and operating systems. Students will be able to describe the relationships between these concepts and components.
Assessments	Students will write sample algorithms, assemble and debug a program that directs the instructor to make a peanut butter & jelly sandwich.
In Class	PB&J
Reading	
Homework	Reflect on PB&J assignment Continue exploring Pokémon

1.3 1.03

Lesson 1.03	<i>String & Console Output</i>
Objectives	Students will correctly assemble a complete program that uses a class header, body, and main method. Students will correctly use print, println, and escape sequences.
Assessments	Students will create a starter Pokémon program Students will complete several questions.
In Class	SC 1.6–9, 1.11–14 LP Welcome
Reading	1.2
Homework	EX 1.1–5

1.4 1.04

Lesson 1.04	<i>Common Errors & Comments</i>
Objectives	Students will create simple programs with comments Students will be able to list and apply the steps necessary for avoiding syntax errors.
Assessments	Students will complete a worksheet (WS 1.4). Students will develop a personal check-list for spotting syntax errors.
In Class	WS 1.4
Reading	1.3
Homework	EX 1.6–9

1.5 1.05

Lesson 1.05	<i>Static Methods & Method Calls</i>
Objectives	Students will use procedural decomposition to plan complex programs using structure diagrams. Students will manage complexity by using method calls.
Assessments	Students will complete problems.
In Class	SC 1.22, 1.23, 1.26, 1.29
Reading	1.4
Homework	EX 1.11, 1.12, 1.14, 1.16

1.6 1.06

Lesson 1.06	<i>Static Methods & Method Calls</i>
Objectives	Students will use structure diagrams to plan complex programs. Students will manage complexity by using method calls.
Assessments	Students will complete problems, students will write a structured Pikachu program.
In Class	LP StarFigures LP PikachuChatter
Reading	1.5
Homework	Outline ch 1 PP 1.1, 1.3

1.7 1.07

Lesson 1.07	<i>Programming Project</i>
Objectives	Students will construct a program containing method calls and static methods.
Assessments	Students will submit a complete, functional program by the end of class.
In Class	PP 1.2, 1.5
Reading	
Homework	Note check (add summaries if needed)

1.8 1.08

Lesson 1.08	<i>Finding & Fixing Errors</i>
Objectives	Students will find errors in their returned homework assignments, and correct their code.
Assessments	Students will re-submit all homework assignments with corrected answers.
In Class	Fix homework
Reading	Review Ch. 1
Homework	Submit questions for review

1.9 1.09

Lesson 1.09	<i>Review</i>
Objectives	Students will identify weaknesses in their Unit 1 knowledge.
Assessments	Students will create a personalized list of review topics to guide tonight's study session.
In Class	Review questions WS 1.9 Practice test
Reading	
Homework	Study

1.10 1.99

Unit 1 Test	<i>Programming & Java</i>
In Class	Test 0 Section I Test 0 Section II

1.11 1.XX

Lesson 1.XX	<i>Open Ended Programming Project(Lesson 1.07 Alternative)</i>
Objectives	Students will be able to ideate and construct a program containing method calls and static methods.
Assessments	Submit a complete, functional program by the end of class
In Class	Check class notes for completion
Reading	
Homework	All students must turn in notes for each day of class

1.12 Abbreviations

- **WS** — Worksheet
- **SC** — Self-Check problem (in the textbook)
- **EX** — Exercise (in the textbook)
- **PP** — Programming Project (in the textbook)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Unit 2: Working with Data & Basic Control Flow (3 weeks)

The following curriculum map is a day-by-day listing of the AP Computer Science course in chronological order. Each row represents one day of class, based on a medium-paced class. Readings from the textbook and homework assignments are included on the day when they should be assigned. Refer to the Introduction document for information about how to adjust this pacing for your specific classroom.

- [Unit 2 Slides](#)
- [Unit 2 Word Bank](#)
- [Curriculum Assets](#)

LP	Title	In Class	Reading	Homework
2.00	Test Review & Reteach	Review test		Test corrections
2.01	Basic Data Concepts	WS 2.1	2.1 except for "Mixing types and Casting"	SC 2.1-2.3 (4th, 5th : 2.1,2.3,2.4)
2.02	Declaring & Assigning Variables	WS 2.2 SC 2.7, 2.11 (4th, 5th: 2.8, 2.13) E 2.1	2.2 up to "String Concatenation"	SC 2.5,2.6,2.9, 2.12-2.15 (4th, 5th: 2.6,2.7,2.1,2.14-2.17)
2.03	String Concatenation & Increment Decrement Operators	Grudgeball	Rest of 2.2	SC 2.4 (4th, 5th: 2.5)
2.04	Mixing Types & Casting	WS 2.4 Poster 2.4	Rest of 2.2	finish WS 2.4
2.05	for Loops	WS 2.5 SC 2.18,2.23, 2.24 (4th, 5th: 2.21,2.26,2.27)	2.3 up to "Nested for Loops"	SC 2.19-2.21 (4th, 5th: 2.22-2.24)
2.06	nested for Loops	SC 2.28-2.30 (4th, 5th: 2.31-2.33), E 2.5	2.3 "Nested for Loops"	SC 2.26, 2.27 (4th, 5th: 2.29, 2.30),E 2.4
2.07	Scope & Pseudocode	WS 2.7 Discuss PP 2.1	2.4 "Scope" and "Pseudocode"	SC 2.31-2.33 (4th, 5th: 2.34-2.36)
2.08	Programming Project	Start PP 2.4	Read 2.4 "Class Constants"	Outline ch 2 (omit 2.5)
2.09	Programming Project	Complete PP 2.4		\[TBD practice question\]
2.10	Finding & Fixing Errors	Fix HW		Submit questions for review

2.11	Review (Review questions)	WS 2.11 practice test	Review ch 2 (omit 2.5)	Study
2.99	(Unit 2 Test)	Test 1 Section I Test 1 Section II		

1.1 2.00

Lesson 2.00	<i>Test Review & Reteach</i>
Objectives	Students will re-learn or strengthen content knowledge and skills from Unit 1.
Assessments	Students will re-submit test answers with updated corrections for partial or full credit, depending on instructor preference.
In Class	Review test
Reading	
Homework	Test corrections

1.2 2.01

Lesson 2.01	<i>Basic Data Concepts</i>
Objectives	Students will be able to identify and categorize data types. Students will identify operators and operands, and will correctly apply rules or precedence.
Assessments	Using operator/operand expression sets, students will use rules of precedence to correctly write code that yields a given answer. Using operator/operand expression sets, students will create their own expressions and predict the output.
In Class	WS 2.1
Reading	2.1 except for "Mixing Types and Casting"
Homework	SC 2.1–3 (4th: 2.1, 2.3, 2.4)

1.3 2.02

Lesson 2.02	<i>Declaring & Assigning Variables</i>
Objectives	Students will be able to identify, declare, and assign variables.
Assessments	Students will write a program that converts temperature from Fahrenheit to Celsius.
In Class	WS 2.2 SC 2.7, 2.11 (4th: 2.8, 2.13) E 2.1
Reading	2.2 up to “ <i>String Concatenation</i> ”
Homework	SC 2.5, 2.6, 2.9, 2.12–15 (4th: 2.6, 2.7, 2.10, 2.14–17)

1.4 2.03

Lesson 2.03	<i>String Concatenation & Increment Decrement Operators</i>
Objectives	Students will apply the rules of string concatenation, students will correctly interpret incrementing and decrementing statements.
Assessments	Students will evaluate statements and predict output during a game of grudgeball.
In Class	Grudgeball
Reading	Rest of 2.2 up to “ <i>Variables and Mixing Types</i> ”
Homework	SC 2.4 (4th: 2.5)

1.5 2.04

**Lesson
2.04*****Mixing Types & Casting***

Objectives	<p>Students will be able to describe which types automatically convert into others when appearing together in expressions, and predict how an expression with mixed types will evaluate.</p> <p>Students will be able to convert types by casting.</p>
Assessments	<p>Students will use “<i>Zombie Rules</i>” of precedence to correctly write code that yields a given answer</p> <p>Students will create their own expressions & predict output by completing and trading worksheets.</p>
In Class	<p>WS 2.4</p> <p>Poster 2.4</p>
Reading	Rest of 2.2
Homework	Finish WS 2.4

1.6 2.05

Lesson 2.05	<i>for Loops</i>
Objectives	<p>Students will trace loops to predict program behavior</p> <p>Students will construct loops to execute simple tasks.</p>
Assessments	Students will trace and construct loops in problems.
In Class	WS 2.5 SC 2.18, 2.23, 2.24 (4th: 2.21, 2.26, 2.27)
Reading	2.3 up to “ <i>Nested for Loops</i> ”
Homework	SC 2.19–21 (4th: 2.22–24)

1.7 2.06

Lesson 2.06	Nested for Loops
Objectives	Students will trace nested loops to predict program behavior Students will construct loops to execute simple tasks.
Assessments	Students will trace and construct nested loops in problems.
In Class	SC 2.28–30 (4th: 2.31–33) E 2.5
Reading	2.3 “Nested for Loops”
Homework	SC 2.26, 2.27 (4th: 2.29, 2.30) E 2.4

1.8 2.07

Lesson 2.07	Scope & Pseudocode
Objectives	Students will be able to identify the scope of a variable and identify common scope errors.
Assessments	Students will complete a worksheet.
In Class	WS 2.7 Discuss PP 2.1
Reading	2.4 “Scope” and “Pseudocode”
Homework	SC 2.31–33 (4th 2.34–36)

1.9 2.08

Lesson 2.08	Programming Project
Objectives	Students will plan and construct a structured program containing nested loops.
Assessments	Students will submit a complete, functional program by the end of next class.
In Class	Start PP 2.4
Reading	Read 2.4 “Class Constants”
Homework	Outline ch 2 (omit 2.5)

1.10 2.09

Lesson 2.09	<i>Programming Project</i>
Objectives	Students will plan and construct a structured program containing nested loops.
Assessments	Students will submit a complete, functional program by the end of next class.
In Class	Complete PP 2.4
Reading	
Homework	\[TBD practice question\]

1.11 2.10

Lesson 2.10	<i>Finding & Fixing Errors</i>
Objectives	Students will find errors in their returned homework assignments, and correct their code.
Assessments	Students will re-submit all homework assignments with corrected answers.
In Class	Fix homework
Reading	
Homework	Submit questions for review

1.12 2.11

Lesson 2.11	Review
Objectives	Students will identify weaknesses in their Unit 1 knowledge.
Assessments	Students will create a personalized list of review topics to guide tonight's study session.
In Class	Review questions WS 2.11 Practice test
Reading	Review ch 2 (omit 2.5)
Homework	Study

1.13 2.99

Unit 2 Test <i>Working with Data & Basic Control Flow</i>	
In Class	Test 1 Section I Test 1 Section II

1.14 Abbreviations

- **WS** — Worksheet
- **SC** — Self-Check problem (in the textbook)
- **EX** — Exercise (in the textbook)
- **PP** — Programming Project (in the textbook)

formatted by [Markdeep 1.093](#) 📝

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Unit 3: Advanced Data & Control Flow (4 weeks)

The following curriculum map is a day-by-day listing of the AP Computer Science course in chronological order. Each row represents one day of class, based on a medium-paced class. Readings from the textbook and homework assignments are included on the day when they should be assigned. Refer to the Introduction document for information about how to adjust this pacing for your specific classroom.

- [Unit 3 Slides](#)
- [Unit 3 Word Bank](#)
- [Curriculum Assets](#)
- [Consumer Review Lab](#)

LP	Title	In Class	Reading	Homework
3.00	Test Review & Reteach	Algorithm for Solving Problems		Test corrections
3.01	Parameters	SC 3.1 – 3.3	3.1 up to "Limitations of Parameters"	SC 3.4-3.7
3.02	Limitations of Parameters & Multiple Parameters	(Art project)	"3.1 "Limitations of Parameters", "Multiple Parameters", ""Parameters versus Constants"""	Jazz up art project and program
3.03	Return Values	SC 3.14 – 3.16	"3.1 "Overloading Methods", 3.2 ""Methods That Return Values"""	SC 3.17, E 3.1
3.04	Programming Project	WS 3.4 Equestria		SC 3.18, 3.19
3.05	Using Objects & String Processing	WS 3.5	3.3 up to "Interactive Programs and Scanner Objects"	SC 3.19-3.21
3.06	Interactive Programs & Scanner Objects	SC 3.24 – 3.26, (5th: 3.23 – 3.25); E 3.12, 3.14, 3.15	3.3 "Interactive programming" and "Sample interactive program"	Outline ch 3 (omit 3.4)
3.07	Pokémon Battle Programming Project	WS 3.7 LP Battle		Summarize notes since last exam
3.08	Finding & Fixing Errors	Fix HW webmaker.org		SC 4.1-4.4
3.09	Relational Operators & if/else	Operator Precedence Grudgeball	4.1 up to "nested if else statements"	SC 4.7-4.9; E 4.1-4.2
3.10	Nested if/else Statements	WS 3.10 Teach mini-lessons SC 4.5, 4.6, E 4.3	4.1 "Nested if/else" and "Flow of control"	E 4.4, 4.5
3.11	Reducing	(Refactoring com	4.1, "Factoring if/else"	Outline ch 4

	Redundancy	petition)	statements" and "Testing multiple conditions"	(omit 4.4, 4.5)
3.12	Cumulative Algorithms	Tally code on board,Collaborative Programming Exercise WS 3.12	Read 4.2	PP 4.2
3.13	while Loops	SC 5.1 – 5.4, E 5.2 WS 3.13	5.1 skip "do/while loops"	E 5.2
3.14	Random Numbers	SC 5.5-5.7; E 5.4, 5.5	5.1 "Random numbers"	PP 5.1
3.15	Fencepost & Sentinel Loops	WS 3.15 Teach mini-lessons	5.2	E 5.6, 5.8
3.16.1	Boolean Logic	SC 5.27, 5.29 WS 3.16 (RPS, Pig) DeMorgan's Law Poster 3.16.1 Poster 3.16.2	5.3	Outline ch 5 (through 5.3)
3.16.2	Boolean Logic (Day 2)			
3.17	Finding & Fixing Errors	(Fix HW)		Submit questions for review
3.18.1	Consumer Review Lab (day 1)	Consumer Review Lab Activity 1	Review ch 3-5	
3.18.2	Consumer Review Lab (day 2)	Consumer Review Lab Activity 2		
3.18.3	Consumer Review Lab (day 3)	Consumer Review Lab Activity 3		
3.18.4	Consumer Review Lab (day 4)	Consumer Review Lab Activity 4		
3.18.5	Consumer	Consumer Review Lab		

	Review Lab (day 5)	Activity 5		
3.18.6	Consumer Review Lab (day 6)	Consumer Review Lab Activity 5 (day 2)		
3.19	Review	(Review questions), WS 3.18 practice test		Study
3.99	Unit 3 test	Test 2 Guide Test 2 Section I Test 2 Section II		
3.XX	Alternative Project: Frac Calc			
3.XX1	Alternative Project: Calculator	work on project	conduct research	Continue working on project

Students are expected to work on project in class. | **Reading** | Students are expected to conduct research. | **Homework** | Continue working on project.

1.1 3.00

Lesson 3.00	<i>Test Review & Reteach</i>
Objectives	Students will re-learn or strengthen content knowledge and skills from Unit 2.
Assessments	Students will re-submit test answers with updated corrections for partial or full credit, depending on instructor preference.
In Class	Algorithm for Solving Problems
Reading	
Homework	Test corrections

1.2 3.01

Lesson 3.01	Parameters
Objectives	Students will correctly construct formal and actual parameters (arguments). Students will predict the output of programs that use parameters.
Assessments	Students will teach a mini-lesson explaining the relationship between parameters and values stored in memory. Students will submit questions.
In Class	SC 3.1–3
Reading	3.1 up to “ <i>Limitations of Parameters</i> ”
Homework	SC 3.4–7

1.3 3.02

Lesson 3.02	Limitations of Parameters & Multiple Parameters
Objectives	Students will modify programs using parameters and class constants to create original artworks.
Assessments	Students will complete an art project and “artist statement” justifying their programming choices.
In Class	Art project
Reading	3.1 “ <i>Limitations of Parameters</i> ”, “ <i>Multiple Parameters</i> ”, “ <i>Parameters versus Constants</i> ”
Homework	Jazz up art project and program

1.4 3.03

Lesson 3.03	Return Values
Objectives	Students will write a program that returns values.
Assessments	Students will complete questions and write a program to meet a Pokémon Challenge.
In Class	SC 3.14–16
Reading	3.1 “ <i>Overloading Methods</i> ” 3.2 “ <i>Methods That Return Values</i> ”
Homework	SC 3.17 E 3.1

1.5 3.04

Lesson 3.04	<i>Programming Project</i>
Objectives	Students will write a program that uses parameters, the math class, and returns values.
Assessments	Students will submit an Equestria program by the end of class.
In Class	WS 3.4 Equestria
Reading	
Homework	SC 3.18–19

1.6 3.05

Lesson 3.05	<i>Using Objects & String Processing</i>
Objectives	Students will be able to differentiate between primitive and object types. Students will apply 0-indexing and string processing techniques to predict the output of a program.
Assessments	Students will complete WS 3.5
In Class	WS 3.5
Reading	3.3 up to “ <i>Interactive Programs and Scanner Objects</i> ”
Homework	SC 3.19–21

1.7 3.06

Lesson 3.06	<i>Interactive Programs & Scanner Objects</i>
Objectives	Students will write programs that accept user input using a scanner object.
Assessments	Students will complete problems.
In Class	SC 3.24–26 E 3.12,14,15
Reading	3.3 “ <i>Interactive Programming</i> ” and “ <i>Sample Interactive Program</i> ”
Homework	Outline ch 3 (omit 3.4)

1.8 3.07

Lesson 3.07	Pokémon Battle Programming Project
Objectives	Students will write a program that requests user input and returns data.
Assessments	Students will write a program that calculates damage done to Pokémon in a battle.
In Class	WS 3.7 LP Battle
Reading	
Homework	Summarize notes since last exam

1.9 3.08

Lesson 3.08	Finding & Fixing Errors
Objectives	Students will find errors and correct their previously submitted homework and classwork assignment.
Assessments	Students will re-submit all homework assignments with corrected answers.
In Class	Fix homework webmaker.org
Reading	
Homework	SC 4.1–4

1.10 3.09

Lesson 3.09	<i>Relational Operators & if/else</i>
Objectives	Students will be able to evaluate relational expressions, predict and trace the flow of an if statement.
Assessments	Students will evaluate relational expressions and practice correct if statement syntax during a game of grudgeball.
In Class	Operator Precedence Grudgeball
Reading	4.1 up to "Nested If/Else Statements"
Homework	SC 4.7–9 E 4.1–2

1.11 3.10

Lesson 3.10	<i>Nested if/else Statements</i>
Objectives	Students will be able to choose which if statements to use for different problems Students will use correct syntax for the different if statements.
Assessments	Students will teach a mini-lesson on sequential or nested if statements. Students will submit several questions.
In Class	WS 3.10 Teach mini-lessons SC 4.5–6 E 4.3
Reading	4.1 "Nested If/Else" and "Flow of Control"
Homework	EX 4.4–5

1.12 3.11

Lesson 3.11	<i>Reducing Redundancy</i>
Objectives	Students will simplify code and reduce redundancy by factoring if/else statements and testing multiple conditions simultaneously.
Assessments	Students will complete a class competition.
In Class	Refactoring competition
Reading	4.1, "Factoring If/Else Statements" and "Testing Multiple Conditions"
Homework	Outline ch 4 (omit 4.4, 4.5)

1.13 3.12

Lesson 3.12	<i>Cumulative Algorithms</i>
Objectives	Students will find and correct syntax errors in conditional cumulative algorithms.
Assessments	Students will write, modify, and correct programs written by others.
In Class	Tally code on board Collaborative Programming Exercise WS 3.12
Reading	4.2
Homework	PP 4.2

1.14 3.13

Lesson 3.13	<i>while Loops</i>
Objectives	Students will trace while loops to predict (1) the number of times the body executes and (2) the output of the code. Students will be able to differentiate between while loops, if statements, and for loops.
Assessments	Students will complete questions.
In Class	SC 5.1–4 E 5.2 WS 3.13
Reading	5.1 (skip "Do/While Loops")
Homework	EX 5.2

1.15 3.14

Lesson 3.14	<i>Random Numbers</i>
Objectives	Students will be able to write expressions that generate a random integer between any two values.
Assessments	Students will complete questions.
In Class	SC 5.5–7 E 5.4–5
Reading	5.1 “ <i>Random Numbers</i> ”
Homework	PP 5.1

1.16 3.15

Lesson 3.15	<i>Fencepost & Sentinel Loops</i>
Objectives	Students will be able to describe when to use fencepost and sentinel loops. Students will use proper syntax to construct these control structures.
Assessments	Students will teach a mini-lesson explaining the relationship between parameters and values stored in memory.
In Class	WS 3.15 Teach mini-lessons
Reading	5.2
Homework	EX 5.6,8

1.17 3.16.1

Lesson 3.16	Boolean Logic (Day 1)
Objectives	Students will work in pairs to write a game that plays Rock Paper Scissors.
Assessments	Students will submit a program at the end of 2 or 3 class periods.
In Class	SC 5.27, 5.29 WS 3.16 (RPS, Pig) DeMorgan's Law Poster 3.16.1 Poster 3.16.2
Reading	5.3
Homework	Outline ch 5 (through 5.3)

1.18 3.16.2

| [Lesson 3.16](#) | Boolean Logic (Day 2) |:-----|:-----|

1.19 3.17

Lesson 3.17	Finding & Fixing Errors
Objectives	Students will find errors in their returned homework assignments, and correct their code.
Assessments	Students will re-submit all homework assignments with corrected answers.
In Class	Fix homework
Reading	
Homework	Submit questions for review

1.20 3.18.1

Lesson 3.18	Consumer Review Lab (Day 1)
Objectives	Students will complete a long-form lab, using string literals, static methods, if statements, while loops, algorithms, and the String class.
Assessments	Students will complete the College Board's AP CS A Consumer Review Lab. Students will answer end of activity Check your understanding and complete Open-ended activity.
In Class	Lab: Consumer Review Lab Consumer Review Lab Activity 1
Reading	Review ch 3-5
Homework	

1.21 3.18.2

Lesson 3.18	Consumer Review Lab (Day 2)
Objectives	
Assessments	
In Class	Consumer Review Lab Activity 2
Reading	
Homework	

1.22 3.18.3

Lesson 3.18	Consumer Review Lab (Day 3)
Objectives	
Assessments	
In Class	Consumer Review Lab Activity 3
Reading	
Homework	

1.23 3.18.4

Lesson 3.18	<i>Consumer Review Lab (Day 4)</i>
Objectives	
Assessments	
In Class	Consumer Review Lab Activity 5
Reading	
Homework	

1.24 3.18.5

Lesson 3.18	<i>Consumer Review Lab (Day 5)</i>
Objectives	
Assessments	
In Class	Consumer Review Lab Activity 5
Reading	
Homework	

1.25 3.18.6

Lesson 3.18	<i>Consumer Review Lab (Day 6)</i>
Objectives	
Assessments	
In Class	Consumer Review Lab Activity 5 (day 2)
Reading	
Homework	

1.26 3.19

Lesson 3.19	Review
Objectives	Students will identify weaknesses in their Unit 3 knowledge.
Assessments	Students will create a personalized list of review topics to guide tonight's study session.
In Class	Review questions WS 3.18 Practice Test
Reading	
Homework	Study

1.27 3.99

Unit 3 Test <i>Advanced Data & Control Flow</i>	
Guide	Test 2 Guide
In Class	Test 2 Section I Test 2 Section II

1.28 3.XX

Unit 3 Alternative Project	<i>Frac Calc</i>
In Class	Frac Calc

1.29 3.XX1

Programming Project (FracCalc Alternative)

Unit 3 Alternative Project

?

Objectives	Students will conduct user-centered research, plan and create, and test, evaluate, and share the end product.
Assessments	Students will submit project for end of Unit 3 assessment.
In Class	Students are expected to work on project in class.
Reading	Students are expected to conduct research
Homework	Continue working on project.

1.30 Abbreviations

- **WS** — Worksheet
- **SC** — Self-Check problem (in the textbook)
- **EX** — Exercise (in the textbook)
- **PP** — Programming Project (in the textbook)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Unit 4: Arrays, Lists, & Files (4 weeks)

The following curriculum map is a Day-by-Day listing of the AP Computer Science course in chronological order. Each row represents one Day of class, based on a medium-paced class. Readings from the textbook and homework assignments are included on the Day when they should be assigned. Refer to the Introduction document for information about how to adjust this pacing for your specific classroom.

- [Unit 4 Slides](#)
- [Unit 4 Word Bank](#)
- [Curriculum Assets](#)
- [Magpie Chatbot Lab](#)

LP	Title	In Class	Reading	Homework
4.00	Test Review & Reteach	Review test	7.1 up to Arrays and Methods	Test corrections
4.01	Array Basics	WS 4.1 Poster 4.2	7.1 For-Each Loop and The Arrays Class	SC 7.1, 7.7, 7.9
4.02	For-Each Loop & Arrays Class	WS 4.2	7.2 up to Reversing an Array	SC 7.12-7.14
4.03	Printing, Searching, & Testing for Equality	WS 4.3 Mini-lessons		SC 7.14-7.17, E 7.3
4.03	Printing, Searching, & Testing for Equality (Day 2)		7.3	SC 7.19-7.21
4.04	Reference Semantics	WS 4.4 7.4 up to Command-Line Arguments	E 7.9, 7.10	
4.05	Shifting Values & Arrays of Objects	SC 7.22, 7.23, 7.25, 7.26, 7.30; E 7.16	7.4 Nested Arrays, 7.5 Rectangular Two Dimensional Arrays	SC 7.27-7.29, E 7.14
4.06	Nested Loop Algorithms & Rectangular Arrays	WS 4.6	10.1 up to Adding to and Removing from an ArrayList	SC?
4.07	ArrayList	Grudgeball Poster 4.7		Outline Ch. 7 and 10.1
4.08	Finding & Fixing Errors	Fix HW	Review Ch. 7, 10.1 for Magpie lab	Submit questions for review
4.09 01	? (Day 1)	Magpie Chatbot Lab Activity 1 & 2	Barron's Ch. 6 (8th or later: Ch. 7)	
4.09 02	? (Day 2)	Magpie Chatbot Lab Activity 2	Barron's Ch. 6 (8th or later: Ch. 7)	

4.09 03	? (Day 3)	Magpie Chatbot Lab Activity 3	Barron's Ch. 6 (8th or later: Ch. 7)	
4.09 04	? (Day 4)	Magpie Chatbot Lab Activity 4		Barron's Ch. 6 (8th or later: Ch. 7)practice questions
4.09 05	? (Day 5)	Magpie Chatbot Lab Activity 5		Check and correct Barron's Ch. 6 (8th or later: Ch. 7) questions
4.09a 01	Steganography Lab (Day 1)	Steganography Lab Activity 1	Barron's Ch. 6 (8th or later: Ch. 7)	
4.09a 02	Steganography Lab (Day 2)	Steganography Lab Activity 2	Barron's Ch. 6 (8th or later: Ch. 7)	
4.09a 03	Steganography Lab (Day 3)	Steganography Lab Activity 3	Barron's Ch. 6 (8th or later: Ch. 7)	
4.09a 04	Steganography Lab (Day 4)	Steganography Lab Activity 4	Barron's Ch. 6 (8th or later: Ch. 7)	
4.09a 05	Steganography Lab (Day 5)	Steganography Lab Activity 5		Barron's Ch. 6 (8th or later: Ch. 7)practice questions
4.09a 06	Steganography Lab (Day 6)	Steganography Lab Activity 5 (Day 2)		Check and correct Barron's Ch. 6 (8th or later: Ch. 7) questions
4.10	Review	Review questions WS 4.10 practice test		Study
4.99	Unit 4 test	Test 3 Section I Test 3 Section II		
4.XX	?(Magpie Alternative)			

Lesson 4.00	Test Review & Reteach
Objectives	Students will re-learn or strengthen content knowledge and skills from Unit 3.
Assessments	Students will re-submit test answers with updated corrections for partial or full credit, depending on instructor preference.
In Class	Review test
Reading	7.1 up to "Arrays and Methods"
Homework	Test corrections

1.2 4.01

Lesson 4.01	Array Basics
Objectives	Students will define, populate, and access arrays.
Assessments	Students will complete manipulatives exercises on WS 4.1.1
In Class	WS 4.1 Poster 4.2
Reading	7.1 "For-Each Loop" and "The Arrays Class"
Homework	SC 7.1,7,9

1.3 4.02

Lesson 4.02	For-Each Loop & Arrays Class
Objectives	Students will define, populate, and access arrays.
Assessments	Students will complete manipulatives exercises on WS 4.2
In Class	WS 4.2
Reading	7.2 up to "Reversing an Array"
Homework	SC 7.12-14

1.4 4.03.1

Lesson 4.03	<i>Printing, Searching, & Testing for Equality (Day 1)</i>
Objectives	Students will be able to manipulate single-dimension arrays using a variety of array transversal algorithms.
Assessments	Students will teach a mini-lesson on printing, searching/replacing, testing for equality, reversing an array, or string traversal. Students will complete a quiz at the end of Day 2.
In Class	WS 4.3 Teach mini-lessons
Reading	
Homework	SC 7.14–17 E 7.3

1.5 4.03.2

Lesson 4.03	<i>Printing, Searching, & Testing for Equality (Day 2)</i>
Objectives	
Assessments	
In Class	
Reading	7.3
Homework	SC 7.19–21

1.6 4.04

Lesson 4.04	Reference Semantics
Objectives	Students will be able to compare and contrast how primitives and arrays are treated when passed as parameters.
Assessments	Students will complete graphic organizers and a worksheet. Some students will complete a Pokémon Challenge for extra credit.
In Class	WS 4.4
Reading	7.4 up to "Command-Line Arguments"
Homework	EX 7.9–10

1.7 4.05

Lesson 4.05	Shifting Values & Arrays of Objects
Objectives	Students will be able to shift elements within an array and construct arrays of objects.
Assessments	Students will complete Practice questions and model memory manipulation using array whiteboards.
In Class	SC 7.22,23,25,26,30 E 7.16
Reading	7.4 "Nested Arrays" 7.5 "Rectangular Two Dimensional Arrays"
Homework	SC 7.27–29 E 7.14

1.8 4.06

Lesson 4.06	Nested Loop Algorithms & Rectangular Arrays
Objectives	Students will correctly adjust nested loop headers for use with arrays Students will correctly construct two-dimensional arrays
Assessments	Students will complete WS 4.6
In Class	WS 4.6
Reading	10.1 up to "Adding to and Removing from an ArrayList"
Homework	SC [_TBD_]

1.9 4.07

Lesson 4.07	ArrayList
Objectives	Students will construct code using ArrayList Students will predict the output of methods that take arrays as parameters and/or return arrays.
Assessments	Students will evaluate statements and predict output during a game of Grudgeball.
In Class	Grudgeball Poster 4.7
Reading	
Homework	Outline Ch. 7 and 10.1

1.10 4.08

Lesson 4.08	Finding & Fixing Errors
Objectives	Students will find errors in their returned homework assignments, and correct their code.
Assessments	Students will re-submit all homework assignments with corrected answers.
In Class	Fix homework
Reading	Review Ch. 7, 10.1 for Magpie lab
Homework	Submit questions for review

1.11 4.09.1

Lesson 4.09	<i>Magpie Lab (Day 1)</i>
Objectives	Students will complete a long-form lab, using if statements, algorithms, the String class, arrays, and ArrayLists.
Assessments	Students will complete the College Board's AP CS A Magpie Chatbot Lab. Students will answer assessment questions on the fourth class exam.
In Class	Lab:? Magpie Chatbot Lab Activity 1 & 2
Reading	Barron's Ch. 6 (8th or later: Ch. 7)
Homework	

1.12 4.09.2

Lesson 4.09	<i>Magpie Lab (Day 2)</i>
Objectives	
Assessments	
In Class	Magpie Chatbot Lab Activity 2
Reading	Barron's Ch. 6 (8th or later: Ch. 7)
Homework	

1.13 4.09.3

Lesson 4.09	<i>Magpie Lab (Day 3)</i>
Objectives	
Assessments	
In Class	Magpie Chatbot Lab Activity 3
Reading	Barron's Ch. 6 (8th or later: Ch. 7)
Homework	

1.14 4.09.4

Lesson 4.09	<i>Magpie Lab (Day 4)</i>
Objectives	
Assessments	
In Class	Magpie Chatbot Lab Activity 4
Reading	
Homework	Barron's Ch. 6 (8th or later: Ch. 7)practice questions

1.15 4.09.5

Lesson 4.09	<i>Magpie Lab (Day 5)</i>
Objectives	
Assessments	
In Class	Magpie Chatbot Lab Activity 5
Reading	
Homework	Check and correct Barron's Ch. 6 (8th or later: Ch. 7) questions

1.16 4.09a.1

Lesson 4.09a	<i>Steganography Lab (Day 1)</i>
Objectives	Students will complete a long-form lab, using arithmetic expressions, methods, if statements, algorithms, while/for loops, arrays, and ArrayLists.
Assessments	Students will complete the College Board's AP CS A Steganography Lab. Students will answer end of activity Check your understanding and open-ended activity.
In Class	Lab: Steganography Lab Steganography Lab Activity 1
Reading	Barron's Ch. 6 (8th or later: Ch. 7)
Homework	

1.17 4.09a.2

Lesson 4.09a	<i>Steganography Lab (Day 2)</i>
Objectives	
Assessments	
In Class	Steganography Lab Activity 2
Reading	Barron's Ch. 6 (8th or later: Ch. 7)
Homework	

1.18 4.09a.3

Lesson 4.09a	<i>Steganography Lab (Day 3)</i>
Objectives	
Assessments	
In Class	Steganography Lab Activity 3
Reading	
Homework	Barron's Ch. 6 (8th or later: Ch. 7)practice questions

1.19 4.09a.4

Lesson 4.09a	<i>Steganography Lab (Day 4)</i>
Objectives	
Assessments	
In Class	Steganography Lab Activity 4
Reading	
Homework	Check and correct Barron's Ch. 6 (8th or later: Ch. 7) questions

1.20 4.09a.5

Lesson 4.09a	<i>Steganography Lab (Day 5)</i>
Objectives	
Assessments	
In Class	Steganography Lab Activity 5
Reading	
Homework	Check and correct Barron's Ch. 6 (8th or later: Ch. 7) questions

1.21 4.09a.6

Lesson 4.09a	<i>Steganography Lab (Day 6)</i>
Objectives	
Assessments	
In Class	Steganography Lab Activity 5 (Day 2)
Reading	
Homework	Check and correct Barron's Ch. 6 (8th or later: Ch. 7) questions

1.22 4.10

Lesson 4.10	<i>Review</i>
Objectives	Students will identify weaknesses in their Unit 4 knowledge.
Assessments	Students will create a personalized list of review topics to guide tonight's study session.
In Class	Review questions WS 4.10 Practice test
Reading	
Homework	Study

1.23 4.99

Unit 4 Test	Arrays, Lists & Files
In Class	Test 3 Section I Test 3 Section II

1.24 4.XX

Lesson 4.XX	Programming Project(Magpie Alternative)
Objectives	Students will be able to conduct user-centered research, plan and create, test evaluate and share
Assessments	Students will apply if-else, String methods to implement a software application and Submit a complete, functional program.
In Class	
Reading	
Homework	Conduct user-centered research to find design opportunities and barriers.

1.25 Abbreviations

- **WS** — Worksheet
- **SC** — Self-Check problem (in the textbook)
- **EX** — Exercise (in the textbook)
- **PP** — Programming Project (in the textbook)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Unit 5: Object-Oriented Programming (4 weeks)

The following curriculum map is a day-by-day listing of the AP Computer Science course in chronological order. Each row represents one day of class, based on a medium-paced class. Readings from the textbook and homework assignments are included on the day when they should be assigned. Refer to the Introduction document for information about how to adjust this pacing for your specific classroom.

- [Unit 5 Slides](#)
- [Unit 5 Word Bank](#)
- [Curriculum Assets](#)
- [Picture Lab](#)

LP	Title	In Class	Reading	Homework
5.00	Test Review & Reteach	Review test	8.1	Test corrections
5.01	Object Oriented Programming	Practice SC 8.1-8.5 WS 5.1.1	8.2 up to "Mutators and Accessors."	
5.02	Object State & Behavior	WS 5.2	8.3 up to "The Keyword this."	SC 8.9-8.11, 8.13-8.16
5.03	Object Initialization: Constructors	WS 5.3.1 WS 5.3.2	8.4	
5.04	Encapsulation	WS 5.4 Mini-lessons		SC 8.22-8.28
5.05	Finding & Fixing Errors	Fix HW	Review Ch. 8 for Picture Lab	Submit questions for review
5.06 01	Picture Lab (day 1)	Picture Lab Activity 1 & 2 Picture Lab		Summarize notes since last exam
5.06 02	Picture Lab (day 2)	Picture Lab Activity 3 & 4, notebook checks		Outline Ch. 8
5.06 03	Picture Lab (day 3)	Picture Lab Activity 5, notebook checks	Read and highlight Barron's Ch. 2, skip this keyword	
5.06 04	Picture Lab (day 4)	Picture Lab Activity 5 & 6, notebook checks		Barron's Ch. 2 exam, skip #20
5.06 05	Picture Lab (day 5)	Picture Lab Activity 6, Barron's checks	Read and highlight Barron's Ch. 5	
5.06 06	Picture Lab (day 6)	Picture Lab Activity 7		SC 8.28, 8.30
5.06 07	Picture Lab (day 7)	Picture Lab Activity 8	8.5	Finish Picture Lab Activity 8
5.06 08	Picture Lab (day 8)	Picture Lab Activity 9		Cont. Picture Lab Activity 9
5.06	Picture Lab (day	Picture Lab Activity 9,		Submit

09	9)	cont.		questions for review
5.06a 01	Data Lab (day 1)	Data Lab Activity 1 Data Lab		Summarize notes since last exam
5.06a 02	Data Lab (day 2)	Data Lab Activity 2, notebook checks		Outline Ch. 8
5.06a 03	Data Lab (day 3)	Data Lab Activity 3, notebook checks	Read and highlight Barron's Ch. 2, skip this keyword	
5.06a 04	Data Lab (day 4)	Data Lab Activity 3 (day 2), notebook checks		Barron's Ch. 2 exam, skip #20
5.06a 05	Data Lab (day 5)	Data Lab Activity 4, Barron's checks	Read and highlight Barron's Ch. 5	
5.06a 06	Data Lab (day 6)	Data Lab Activity 4 (day 2)		SC 8.28, 8.30
5.06a 07	Data Lab (day 7)	Data Lab Activity 4 (day 3)	8.5	Finish Data Lab Activity 8
5.06a 08	Data Lab (day 8)	Data Lab Activity 4 (day 4)		Cont. Data Lab Activity 9
5.07	Review	Review question WS 5.7 Test practice		Study
5.99	(Unit 5 test)	Test 4 Section I Test 4 Section II		
5.XX	PictureLab Alternative			

1.1 5.00

Lesson 5.00	Test Review & Reteach
Objectives	Students will re-learn or strengthen content knowledge and skills from Unit 4
Assessments	Students will re-submit test answers with updated corrections for partial or full credit, depending on instructor preference.
In Class	Review test
Reading	8.1
Homework	Test corrections

1.2 5.01

Lesson 5.01	<i>Object Oriented Programming</i>
Objectives	Students will be able to describe the relationship between classes, objects, and client code. Students will be able to predict the output of the code that uses objects.
Assessments	Students will complete Practice questions.
In Class	Practice SC 8.1–5 WS 5.1.1
Reading	8.2 up to “ <i>Mutators and Accessors</i> ”
Homework	

1.3 5.02

Lesson 5.02	<i>Object State & Behavior</i>
Objectives	Students will be able to describe classes, objects, and client code. Students will be able to predict the output of the code that uses objects.
Assessments	Students will complete WS 5.2 individually or in pairs.
In Class	WS 5.2
Reading	8.3 up to “ <i>The Keyword this</i> ”
Homework	SC 8.9–11,13–16

1.4 5.03

Lesson 5.03	<i>Object Initialization: Constructors</i>
Objectives	Students will be able to describe and create classes, objects, and client code. Students will be able to predict the output of the code that uses objects.
Assessments	Students will complete Practice questions.
In Class	WS 5.3.1 WS 5.3.2
Reading	8.4
Homework	

1.5 5.04

Lesson 5.04	<i>Encapsulation</i>
Objectives	Students will be able to manipulate single-dimension arrays using a variety of array transversal algorithms.
Assessments	Students will teach a mini-lesson on printing, searching/replacing, testing for equality, reversing an array, or string traversal. Students will complete a quiz at the end of Day 2.
In Class	WS 5.4 Teach mini-lessons
Reading	
Homework	SC 8.22–28

1.6 5.05

**Lesson
5.05****Finding & Fixing Errors**

Objectives	Students will find errors in their returned homework assignments, and correct their code.
Assessments	Students will re-submit all homework assignments with corrected answers.
In Class	Fix homework
Reading	Review Ch. 8 for Picture Lab
Homework	Submit questions for review

1.7 5.06.1

Lesson 5.06	Picture Lab (Day 1)
Objectives	Students will complete a long-form lab, using two dimensional arrays of objects, array traversing algorithms, program analysis, binary numbers, and inheritance.
Assessments	Picture Lab
In Class	Picture Lab Activity 1 & 2 Picture Lab
Reading	
Homework	Summarize notes since last exam

1.8 5.06.2

Lesson 5.06	Picture Lab (Day 2)
Objectives	
Assessments	
In Class	Picture Lab Activity 3 & 4 Notebook checks
Reading	
Homework	Outline Ch. 8

1.9 5.06.3

Lesson 5.06	<i>Picture Lab (Day 3)</i>
Objectives	
Assessments	
In Class	Picture Lab Activity 5 Notebook checks
Reading	Read and highlight Barron's Ch. 2, skip this keyword

| Homework

1.10 5.06.4

Lesson 5.06	<i>Picture Lab (Day 4)</i>
Objectives	
Assessments	
In Class	Picture Lab Activity 5 & 6 Notebook checks
Reading	
Homework	Barron's Ch. 2 exam (skip #20)

1.11 5.06.5

Lesson 5.06	<i>Picture Lab (Day 5)</i>
Objectives	
Assessments	
In Class	Picture Lab Activity 6 Barron's checks
Reading	Read and highlight Barron's Ch. 5
Homework	

1.12 5.06.6

Lesson 5.06	<i>Picture Lab (Day 6)</i>
Objectives	
Assessments	
In Class	Picture Lab Activity 7
Reading	
Homework	SC 8.28,30

1.13 5.06.7

Lesson 5.06	<i>Picture Lab (Day 7)</i>
Objectives	
Assessments	
In Class	Picture Lab Activity 8
Reading	8.5
Homework	Finish Picture Lab Activity 8

1.14 5.06.8

Lesson 5.06	<i>Picture Lab (Day 8)</i>
Objectives	
Assessments	
In Class	Picture Lab Activity 9
Reading	
Homework	Cont. Picture Lab Activity 9

1.15 5.06.9

Lesson 5.06		<i>Picture Lab (Day 9)</i>
Objectives		
Assessments		
In Class		Picture Lab Activity 9, cont.
Reading		
Homework		Submit questions for review

1.16 5.06a.1

Lesson 5.06a	<i>Data Lab (Day 1)</i>
Objectives	Students will complete a long-form lab, using classes, objects, two dimensional arrays of objects, array traversing algorithms, program analysis and while/for loops.
Assessments	Students will complete the College Board's AP CS A Data Lab. Students will answer end of activity Check your understanding and open-ended activity.
In Class	Data Lab Activity 1 Data Lab
Reading	
Homework	Summarize notes since last exam

1.17 5.06a.2

Lesson 5.06a	<i>Data Lab (Day 2)</i>
Objectives	
Assessments	
In Class	Data Lab Activity 2 Notebook checks
Reading	
Homework	Outline Ch. 8

1.18 5.06a.3

Lesson 5.06a	Data Lab (Day 3)
Objectives	
Assessments	
In Class	Data Lab Activity 3 Notebook checks
Reading	Read and highlight Barron's Ch. 2, skip this keyword

| **Homework**

1.19 5.06a.4

Lesson 5.06a	Data Lab (Day 4)
Objectives	
Assessments	
In Class	Data Lab Activity 3 (day 2) Notebook checks
Reading	
Homework	Barron's Ch. 2 exam (skip #20)

1.20 5.06a.5

Lesson 5.06a	Data Lab (Day 5)
Objectives	
Assessments	
In Class	Data Lab Activity 4 Barron's checks
Reading	Read and highlight Barron's Ch. 5
Homework	

1.21 5.06a.6

Lesson 5.06a	Data Lab (Day 6)
Objectives	
Assessments	
In Class	Data Lab Activity 4 (day 2)
Reading	
Homework	SC 8.28,30

1.22 5.06a.7

Lesson 5.06a	Data Lab (Day 7)
Objectives	
Assessments	
In Class	Data Lab Activity 4 (day 3)
Reading	8.5
Homework	Finish Data Lab Activity 8

1.23 5.06a.8

Lesson 5.06a	Data Lab (Day 8)
Objectives	
Assessments	
In Class	Data Lab Activity 4 (day 4)
Reading	
Homework	Cont. Data Lab Activity 9

1.24 5.07

Lesson 5.07	Review
Objectives	Students will identify weaknesses in their Unit 5 knowledge.
Assessments	Students will create a personalized list of review topics to guide tonight's study session.
In Class	Review questions WS 5.7 Test practice
Reading	
Homework	Study

1.25 5.99

Unit 5 Test		<i>Object Oriented Programming</i>
In Class		Test 4 Section I Test 4 Section II

1.26 5.XX

Lesson 5.XX	<i>Programming Project(PictureLab Alternative)</i>
Objectives	Students will be able to conduct user-centered research, plan and create, test, evaluate and share.
Assessments	Apply 2-dimensional arrays, traversal, binary representations of data and submit a complete functional program.
In Class	Project Design
Reading	
Homework	Conduct research work(survey or interviews) and communicating with end-user

1.27 Abbreviations

- **WS** — Worksheet
- **SC** — Self-Check problem (in the textbook)
- **EX** — Exercise (in the textbook)
- **PP** — Programming Project (in the textbook)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Unit 6: Inheritance & Polymorphism (4 weeks)

The following curriculum map is a day-by-day listing of the AP Computer Science course in chronological order. Each row represents one day of class, based on a medium-paced class. Readings from the textbook and homework assignments are included on the day when they should be assigned. Refer to the Introduction document for information about how to adjust this pacing for your specific classroom.

- [Unit 6 Slides](#)
- [Unit 6 Word Bank](#)
- [Curriculum Assets](#)
- [Text Excel](#)

LP	Title	In Class	Reading	Homework
6.00	Test Review & Reteach	(Review test)	9.1	Test corrections
6.01 01	Inheritance Basics (day 1)	WS 6.1 Start class poster Example 6.1	"9.2 up to ""Dividend Stock Behavior"""	Collect images
6.01 02	Inheritance Basics (day 2)	Finish class poster, discuss		
6.02	Overriding Methods & Accessing Inherited Code	WS 6.2	Rest of 9.2 starting from "The Object Class."	
6.03	Interacting with the Object Superclass	Practice SC 9.3, 9.4, 9.9, 9.10, E 9.4; WS 6.3 Poster 6.3	9.3 up to "Interpreting Inheritance Code."	
6.04	Polymorphism	WS 6.4.1 WS 6.4.2 SC 9.11-9.17	Rest of 9.4 "Is-a Versus Has-a Relationships."	SC 9.18, 9.20
6.05	Has-a Relationships	WS 6.5 Value Meal exercise?	9.5 (Optional if covering Interfaces)	
6.06	Interfaces (optional)	? Poster 6.6	9.6 (Optional if covering Abstract classes)	Generate own class hierarchy like Financial hierarchy in book
6.07 01	Programming project (day 1)	PP 9.1, notebook checks		Outline Ch. 9
6.07 02	Programming project (day 2)	PP 9.1, outline checks	Read and outline Barron's	

			Ch. 4 (8th or later: Ch. 5)	
6.07 03	Programming project (day 3)	PP 9.3		Barron's Ch. 4 (8th or later: Ch. 5) exam, self-grade
6.07 04	Programming project (day 4)	E 9.8	Read and outline Barron's Ch. 3 (8th or later: Ch. 4)	
6.07 05	Programming project (day 5)	Barron's Ch. 3 (8th or later: Ch. 4) exam, outline checks	Review Ch. 9	Submit questions for review
6.07a 01	Celebrity Lab (day 1)	Celebrity Lab Activity 1, notebook checks		Outline Ch. 9
6.07a 02	Celebrity Lab (day 2)	Celebrity Lab Activity 2, outline checks	Read and outline Barron's Ch. 4 (8th or later: Ch. 5)	
6.07a 03	Celebrity Lab (day 3)	Celebrity Lab Activity 3		Barron's Ch. 4 (8th or later: Ch. 5) exam, self-grade
6.07a 04	Celebrity Lab (day 4)	Celebrity Lab Activity 4	Read and outline Barron's Ch. 3 (8th or later: Ch. 4)	
6.07a 05	Celebrity Lab (day 5)	Celebrity Lab Activity 5	Review Ch. 9	Submit questions for review
6.07a 06	Celebrity Lab (day 5)	Celebrity Lab Activity 5 (day 2)	Review Ch. 9	Submit questions for review
6.07a 07	Celebrity Lab (day 5)	Celebrity Lab Activity 5 (day 3), outline checks	Review Ch. 9	Submit questions for review
6.08	Finding & Fixing Errors	(Fix HW)	Review Ch. 9	Submit questions for review
6.09	Review	Review questions WS 6.5 Test practice		Study

6.99	Unit 6 test	Test 5 Guide Test 5 Section I Test 5 Section II	
6.XX	?	Text Excel Student Guide A Text Excel Student Guide B Text Excel Student Guide C Text Excel Teacher Guide	

1.1 6.00

Lesson 6.00	<i>Test Review & Reteach</i>
Objectives	Students will re-learn or strengthen content knowledge and skills from Unit 5.
Assessments	Students will re-submit test answers with updated corrections for partial or full credit, depending on instructor preference.
In Class	Review test
Reading	9.1
Homework	Test corrections

1.2 6.01.1

Lesson 6.01	<i>Inheritance Basics (Day 1)</i>
Objectives	Students will correctly define inheritance Students will use proper syntax to extend a class. Students will illustrate is-a relationships. Students will properly implement constructors of derived classes using super.
Assessments	Students will complete a Class Hierarchy Poster as indicated in WS 6.1.
In Class	WS 6.1 Start class poster Example 6.1
Reading	9.2 up to "Dividend Stock Behavior"
Homework	Collect images

1.3 6.01.2

Lesson 6.01	Inheritance Basics (Day 2)
Objectives	
Assessments	
In Class	Finish class poster Discussion
Reading	
Homework	

1.4 6.02

Lesson 6.02	Overriding Methods & Accessing Inherited Code
Objectives	Students will replace superclass behavior by writing overriding methods in the subclass. Students will write subclass methods that access superclass methods.
Assessments	Students will add code to their Class Posters from the previous lesson.
In Class	WS 6.2
Reading	Rest of 9.2 starting from “ <i>The Object Class</i> ”
Homework	

1.5 6.03

Lesson 6.03	Interacting with the Object Superclass
Objectives	Students will replace superclass behavior by writing overriding methods in the subclass. Students will write subclass methods that access superclass methods.
Assessments	Students will complete Practice questions Students will complete a worksheet.
In Class	Practice SC 9.3-4,9-10 E 9.4 WS 6.3 Poster 6.3
Reading	9.3 up to “ <i>Interpreting Inheritance Code</i> ”
Homework	

1.6 6.04

Lesson 6.04	<i>Polymorphism</i>
Objectives	Students will define polymorphism. Students will trace the execution of methods through a class hierarchy and predict output.
Assessments	Students will complete a Tracing Inheritance guide and complete worksheet 6.4.
In Class	WS 6.4.1 WS 6.4.2 SC 9.11-17
Reading	Rest of 9.4 "Is-a Versus Has-a Relationships"
Homework	SC 9.18,20

1.7 6.05

Lesson 6.05	<i>Has-a Relationships</i>
Objectives	Students will be able to identify and explain why two classes have an is-a or a has-a relationship. Students will be able to create a has-a relationship between two classes.
Assessments	Students will complete an AP Section II question "Trio"
In Class	WS 6.5 ValueMeal exercise
Reading	9.5 (Optional if covering Interfaces)
Homework	

1.8 6.06

Lesson 6.06	<i>Interfaces (Optional)</i>
Objectives	Students will implement and use interfaces.
Assessments	Students will complete an in-class competition.
In Class	Interface examples Poster 6.6
Reading	9.6 (Optional if covering Abstract classes)
Homework	Generate own class hierarchy like Financial hierarchy in book

1.9 6.07.1

Lesson 6.07	<i>Programming project (Day 1)</i>
Objectives	Students will write complex code that uses polymorphism, and inheritance.
Assessments	Students will submit a program electronically.
In Class	PP 9.1 Notebook checks
Reading	
Homework	Outline Ch. 9

1.10 6.07.2

Lesson 6.07	<i>Programming project (Day 2)</i>
Objectives	
Assessments	
In Class	PP 9.1 Outline checks
Reading	Read and outline Barron's Ch. 4 (8th or later: Ch. 5)
Homework	

1.11 6.07.3

Lesson 6.07	<i>Programming project (Day 3)</i>
Objectives	
Assessments	
In Class	PP 9.3
Reading	
Homework	Barron's Ch. 4 (8th or later: Ch. 5) exam, self-grade

1.12 6.07.4

Lesson 6.07	Programming project (Day 4)
Objectives	
Assessments	
In Class	EX 9.8
Reading	Read and outline Barron's Ch. 3 (8th or later: Ch. 4)
Homework	

1.13 6.07.5

Lesson 6.07	Programming project (Day 5)
Objectives	
Assessments	
In Class	Barron's Ch. 3 (8th or later: Ch. 4) exam, Outline checks
Reading	Review Ch. 9
Homework	Submit questions for review

1.14 6.07a.1

Lesson 6.07a	Celebrity Lab (Day 1)_
Objectives	Students will complete a long-form lab, using classes, objects, two dimensional arrays of objects, array traversing algorithms, program analysis, while/for loops.
Assessments	Students will complete the College Board's AP CS A Celebrity Lab. Students will answer end of activity Check your understanding and open-ended activity.
In Class	Celebrity Lab Activity 1 Notebook checks
Reading	
Homework	Outline Ch. 9

1.15 6.07a.2

Lesson 6.07a	Celebrity Lab (Day 2)_
Objectives	
Assessments	
In Class	Celebrity Lab Activity 2 Outline checks
Reading	Read and outline Barron's Ch. 4 (8th or later: Ch. 5)
Homework	

1.16 6.07a.3

Lesson 6.07a	Celebrity Lab (Day 3)_
Objectives	
Assessments	
In Class	Celebrity Lab Activity 3
Reading	
Homework	Barron's Ch. 4 (8th or later: Ch. 5) exam, self-grade

1.17 6.07a.4

Lesson 6.07a	Celebrity Lab (Day 4)_
Objectives	
Assessments	
In Class	Celebrity Lab Activity 4
Reading	Read and outline Barron's Ch. 3 (8th or later: Ch. 4)
Homework	

1.18 6.07a.5

Lesson 6.07a	Celebrity Lab (Day 5)_
Objectives	
Assessments	
In Class	Celebrity Lab Activity 5, Outline checks
Reading	Review Ch. 9
Homework	

1.19 6.07a.6

Lesson 6.07a	Celebrity Lab (Day 6)_
Objectives	
Assessments	
In Class	Celebrity Lab Activity 5 (day 2), Outline checks
Reading	Review Ch. 9
Homework	

1.20 6.07a.7

Lesson 6.07a	Celebrity Lab (Day 7)_
Objectives	
Assessments	
In Class	Celebrity Lab Activity 5 (day 3), Outline checks
Reading	Review Ch. 9
Homework	Submit questions for review

1.21 6.08

Lesson 6.08	Finding & Fixing Errors
Objectives	Students will find errors in their returned homework assignments, and correct their code.
Assessments	Students will re-submit all homework assignments with corrected answers.
In Class	Fix homework
Reading	Review Ch. 9
Homework	Submit questions for review

1.22 6.09

Lesson 6.09	Review
Objectives	Students will identify weaknesses in their Unit 6 knowledge.
Assessments	Students will create a personalized list of review topics to guide tonight's study session.
In Class	Review questions WS 6.5 Test practice
Reading	
Homework	Study

1.23 6.99

Unit 6 Test	Inheritance and Polymorphism
Guide	Test 5 Guide
In Class	Test 5 Section I Test 5 Section II

1.24 6.XX

**Unit 6
Project*****Text Excel*****In Class**

? [Text Excel Student Guide A](#) [Text Excel Student Guide B](#) [Text Excel Student Guide C](#) [Text Excel Teacher Guide](#)

1.25 Abbreviations

- **WS** — Worksheet
- **SC** — Self-Check problem (in the textbook)
- **EX** — Exercise (in the textbook)
- **PP** — Programming Project (in the textbook)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Unit 7: Searching & Sorting (3 weeks)

The following curriculum map is a day-by-day listing of the AP Computer Science course in chronological order. Each row represents one day of class, based on a medium-paced class. Readings from the textbook and homework assignments are included on the day when they should be assigned. Refer to the Introduction document for information about how to adjust this pacing for your specific classroom.

- [Unit 7 Slides](#)
- [Unit 7 Word Bank](#)
- [Curriculum Assets](#)
- [Elevens Lab](#)

LP	Title	In Class	Reading	Homework
7.00	Test Review & Reteach	Review test	13.1 up to "Sorting"	Test corrections
7.01	Searching Algorithms	(CS Unplugged Battleship) WS 7.1	13.1 "Sorting"	SC 2, E 13.1-13.3
7.02	Sorting Algorithms		13.1 "Shuffling"	
7.03 01	? (day 1)	Elevens Lab Activity 1	"13.3 skip ""Recursive Binary Search"""	SC 13.16-13.18, 13.21, 13.22
7.03 02	? (day 2)	Elevens Lab Activity 2 (begin)		
7.03 03	? (day 3)	Elevens Lab Activity 2 (end)		Summarize notes since last exam
7.03 04	? (day 4)	Elevens Lab Activity 3 (begin), notebook checks		Outline Ch. 13
7.03 05	? (day 5)	Elevens Lab Activity 3 (end), notebook checks	Read and outline Barron's Ch. 8	
7.03 06	? (day 6)	Elevens Lab Activity 4		Barron's Ch. 8 exam, self-grade
7.03 07	? (day 7)	Elevens Lab Activity 5 (begin), Barron's checks		
7.03 08	? (day 8)	Elevens Lab Activity 5 (end)		
7.03 09	? (day 9)	Elevens Lab Activity 6		(Fix HW)
7.03 10	? (day 10)	Elevens Lab Activity 7		(Fix HW)
7.03 11	? (day 11)	Elevens Lab Activity 8, re-grade fixed HW		
7.03 12	? (day 12)	Elevens Lab Activity 9 (begin), re-grade fixed HW		
7.03	? (day 13)	Elevens Lab Activity 9		Submit questions

13		(end), re-grade fixed HW		for review
7.03 14	? (day 14)	Elevens Lab Activity 10, re-grade fixed HW		
7.03 15	? (day 15)	Elevens Lab Activity 11 (begin)		
7.03 16	? (day 16)	Elevens Lab Activity 11 (end)		
7.04	Review	Review questions		Study
7.99	Unit 7 test	Test 6 Guide Test 6 Section I Test 6 Section II		

1.1 7.00

Lesson 7.00	Test Review & Reteach
Objectives	Students will re-learn or strengthen content knowledge and skills from Unit 6.
Assessments	Students will re-submit test answers with updated corrections for partial or full credit, depending on instructor preference.
In Class	Review test
Reading	13.1 up to “Sorting”
Homework	Test corrections

1.2 7.01

Lesson 7.01	Searching Algorithms
Objectives	Students will compare and contrast the different search algorithms.
Assessments	Students will complete some short answer questions.
In Class	CS Unplugged Battleship WS 7.1
Reading	13.1 “Sorting”
Homework	SC 13.4–6 E 13.1–3

1.3 7.02

Lesson 7.02	<i>Sorting Algorithms</i>
Objectives	Students will compare and contrast different sorting methods and evaluate their relative speed and efficiency.
Assessments	Students will complete some short answer questions on worksheets.
In Class	
Reading	13.1 “ <i>Shuffling</i> ”
Homework	

1.4 7.03.1

Lesson 7.03	<i>Elevens lab (Day 1)</i>
Objectives	Students will complete a long-form lab, demonstrating effective use of object oriented program design, program implementation and analysis, and standard data structures and algorithms.
Assessments	Elevens Lab
In Class	Elevens Lab Activity 1
Reading	13.3 (skip “ <i>Recursive Binary Search</i> ”)
Homework	SC 13.16–21,23–24

1.5 7.03.2

Lesson 7.03	<i>Elevens lab (Day 2)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 2 (begin)
Reading	
Homework	

1.6 7.03.3

Lesson 7.03	<i>Elevens lab (Day 3)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 2 (end)
Reading	
Homework	Summarize notes since last exam

1.7 7.03.4

Lesson 7.03	<i>Elevens lab (Day 4)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 3 (begin) Notebook checks
Reading	
Homework	Outline Ch. 13

1.8 7.03.5

Lesson 7.03	<i>Elevens lab (Day 5)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 3 (end) Notebook checks
Reading	Read and outline Barron's Ch. 8
Homework	

1.9 7.03.6

Lesson 7.03	<i>Elevens lab (Day 6)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 4
Reading	
Homework	Barron's Ch. 8 exam, self-grade

1.10 7.03.7

Lesson 7.03	<i>Elevens lab (Day 7)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 5 (begin) Barron's checks
Reading	
Homework	

1.11 7.03.8

Lesson 7.03	<i>Elevens lab (Day 8)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 5 (end)
Reading	
Homework	

1.12 7.03.9

Lesson 7.03	<i>Elevens lab (Day 9)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 6
Reading	
Homework	Fix homework

1.13 7.03.10

Lesson 7.03	<i>Elevens lab (Day 10)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 7
Reading	
Homework	Fix homework

1.14 7.03.11

Lesson 7.03	<i>Elevens lab (Day 11)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 8 Re-grade fixed homework
Reading	
Homework	

1.15 7.03.12

Lesson 7.03	<i>Elevens lab (Day 12)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 9 (begin) Re-grade fixed homework
Reading	
Homework	

1.16 7.03.13

Lesson 7.03	<i>Elevens lab (Day 13)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 9 (end) Re-grade fixed homework
Reading	
Homework	Submit questions for review

1.17 7.03.14

Lesson 7.03	<i>Elevens lab (Day 14)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 10 Re-grade fixed homework
Reading	
Homework	

1.18 7.03.15

Lesson 7.03	<i>Elevens lab (Day 15)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 11 (begin)
Reading	
Homework	

1.19 7.03.16

Lesson 7.03	<i>Elevens lab (Day 16)</i>
Objectives	
Assessments	
In Class	Elevens Lab Activity 11 (end)
Reading	
Homework	

1.20 7.04

Lesson 7.04	Review
Objectives	Students will identify weaknesses in their Unit 7 knowledge.
Assessments	Students will create a personalized list of review topics to guide tonight's study session.
In Class	Review questions
Reading	
Homework	Study

1.21 7.99

Unit 7 Test	<i>Searching & Sorting</i>
Guide	Test 6 Guide
In Class	Test 6 Section I Test 6 Section II

1.22 Abbreviations

- **WS** — Worksheet
- **SC** — Self-Check problem (in the textbook)
- **EX** — Exercise (in the textbook)
- **PP** — Programming Project (in the textbook)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Unit 8: Recursion (2 weeks)

The following curriculum map is a day-by-day listing of the AP Computer Science course in chronological order. Each row represents one day of class, based on a medium-paced class. Readings from the textbook and homework assignments are included on the day when they should be assigned. Refer to the Introduction document for information about how to adjust this pacing for your specific classroom.

- [Unit 8 Slides](#)
- [Unit 8 Word Bank](#)
- [Curriculum Assets](#)

LP	Title	In Class	Reading	Homework
8.00	Test Review & Reteach	Review test	12.1 up to "Structure of recursive solutions"	Test corrections
8.01	Thinking Recursively	Tower of Hanoi game	Rest of 12.1	
8.02	Writing Recursive Solutions	Grudgeball SC 12.1 – 12.4	12.2	SC 12.5, 12.7-12.9, E 12.1
8.03	Mechanics of Recursion	WS 8.3 Teacher Demo 8.3	13.4?	SC 12.6, 12.10, E 12.3
8.04	MergeSort	Implement mergeSort		SC 13.27-13.30 Notebook Check
8.05	Finding & Fixing Errors	Fix HW	Review Ch. 12.1, 12.2	Submit questions for review
8.06	Review		Study	
8.07	Quiz	Quiz 8.5	Barron's Ch. 7 (8th or later: Ch. 8)	
8.08	Quiz Review & Reteach	Review quiz		Barron's Ch. 7 (8th or later: Ch. 8)

1.1 8.00

Lesson 8.00	<i>Test Review & Reteach</i>
Objectives	Students will re-learn or strengthen content knowledge and skills from Unit 7.
Assessments	Students will re-submit test answers with updated corrections for partial or full credit, depending on instructor preference.
In Class	Review test
Reading	12.1 up to “ <i>Structure of Recursive Solutions</i> ”
Homework	Test corrections

1.2 8.01

Lesson 8.01	<i>Thinking Recursively</i>
Objectives	Students will be able to define recursion.
Assessments	Students will describe recursive methods and compare iterative and recursive methods during a class discussion.
In Class	Tower of Hanoi game
Reading	Rest of 12.1
Homework	

1.3 8.02

Lesson 8.02	<i>Writing Recursive Solutions</i>
Objectives	Students will be able to identify recursive methods and predict the output (or return value) of recursive methods.
Assessments	Students will evaluate statements and predict output during a game of Grudgeball.
In Class	Grudgeball SC 12.1–4
Reading	12.2
Homework	SC 12.5,7–9 E 12.1

1.4 8.03

Lesson 8.03	<i>Mechanics of Recursion</i>
Objectives	Students will be able to model how recursive methods execute.
Assessments	Students will write a recursive method, then model the execution of that method for the instructor. Students will also model a method written by their peers.
In Class	WS 8.3 Teacher Demo 8.3
Reading	13.4
Homework	SC 12.6,10 E 12.3

1.5 8.04

Lesson 8.04	<i>MergeSort</i>
Objectives	Students will use mergeSort to sort an ArrayList.
Assessments	Students will be able to use recursion to sort a list.
In Class	Implement mergeSort
Reading	
Homework	SC 13.27–30 Notebook Check

1.6 8.05

Lesson 8.05	Finding & Fixing Errors
Objectives	Students will find errors in their returned homework and classwork.
Assessments	Students will re-submit all homework and classwork assignments with corrected answers.
In Class	Fix homework
Reading	Review Ch. 12.1–2
Homework	Submit questions for review

1.7 8.06

Lesson 8.06	Review
Objectives	Students will identify weaknesses in their Unit 8 knowledge.
Assessments	Students will create a personalized list of review topics to guide tonight's study session.
In Class	Review Questions
Reading	
Homework	Study

1.8 8.07

Lesson 8.07	Review & Quiz
In Class	Quiz 8.5
Reading	Barron's Ch. 7 (8th or later: Ch. 8)

1.9 8.08

Lesson 8.08	Quiz Review & Reteach
Objectives	Students will re-learn or strengthen content knowledge and skills from Unit 8.
Assessments	Re-submit quiz answers with updated corrections for partial or full credit.
In Class	Review quiz
Reading	
Homework	Barron's Ch. 7 (8th or later: Ch. 8)

1.10 Abbreviations

- **WS** — Worksheet
- **SC** — Self-Check problem (in the textbook)
- **EX** — Exercise (in the textbook)
- **PP** — Programming Project (in the textbook)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Unit 9: AP Test Review (3 weeks)

- Curriculum Assets

1.1 9.00

Lesson 9.00	<i>Reviewing for the AP Exam</i>
Objectives	Students will review and prepare for the AP Computer Science A exam.
Assessments	Bellevue Mastery Tests AP CS 2012 Section II
In Class	
Reading	
Homework	

1.2 9.01

Lesson 9.01	<i>Mock AP exam</i>
Objectives	Students will participate in a mock AP exam.
Assessments	
In Class	
Reading	
Homework	

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Unit 10: Post-AP Exam Projects (4–5 weeks)

We currently have two different curriculum options for after the AP exam, each of which is a self-contained website with lessons, labs and other supporting resources:

- Character Clash
- SpaceBattleArena
- TEALS Minecraft Modding

formatted by [Markdeep 1.093](#) ↗

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 1.02 – Algorithms & Computational Thinking

1.1 Overview

1.1.1 Objectives – Students will be able to...

- **Define** algorithms, programs, hardware, software, and operating systems.
- **Describe** the relationships between these concepts and components.

1.1.2 Assessments – Students will...

- **Write** sample algorithms
- **Assemble and debug** a program that directs the instructor to make a sandwich

1.1.3 Homework – Students will...

1.2 Materials & Prep

- **Projector and computer OR whiteboard and marker**
- **Food items** for peanut butter and jelly sandwich
- **Utensils** for sandwich assembly (spoon for jelly, knife for spreading)
- **Wet wipes or water** to clean hands

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction to vocabulary	10min
Explaining activity, assigning pairs	2min
Activity Round One	10min
Activity Round Two	10–15min
Full-class discussion of debugging methods	5min
HW distribution & exit tickets	2min

1.4 Procedure

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction to Vocabulary [10 minutes]

In this lesson, you will be using yourself as the metaphor to introduce fundamental vocabulary in CS. As a hook, you should leave all of the food prep materials conspicuously laid out in the front of the classroom, without any explanation.

1. You should begin your lesson with an informal, whole-group conversation about humans being the first “computers.” This is a great opportunity to use history, women’s accomplishments in CS, and humor to offer up and drill vocabulary without onerous repetition. Some suggestions for your opening discussion:
 - Computer was originally a job description, and was first used in 1613 to describe someone who performs mathematical calculations.
 - Early computers were mostly women:



http://www.officemuseum.com/IMagesWWW/Eary_1920s_Veterans_Bureau_Calculating_WWI_Vet_Bonuses_LOC.JPG

- Human computers have many similarities to mechanical computers. See how many of these your students can predict with minimal prompting:
 - Brain = CPU and processing
 - Input = information (data) from sensing the environment
 - Sensors = eyes, ears, nose, mouth, fingers/skin
 - Output = behavior, action
- You can model errors and exceptions by dramatically narrating and acting out a confusing set of directions and the resultant mistake. Depending on your personal style, this can be a pratfall, embarrassing social blunder (real or imagined), or simple spelling mistake (input: sounding out a word with silent letters, error/output: the misspelled word)

2. Introduce the definition of an **algorithm**, and invite the students to write an algorithm for you, the computer, to make a peanut butter and jelly sandwich.

- An algorithm may be defined as “a process or set of rules to be followed in calculations or other problem-solving operations”

1.4.3 Explaining Activity and Assigning Pairs (2 minutes)

Give student pairs 5 minutes to write a peanut-butter-and-jelly algorithm. Specify that they should write a **complete set of directions** to describe the process of making the sandwich.

1.4.4 Activity Round 1 (10 minutes)

1. Ask for a student volunteer to come to the front of the classroom to “be Java.”
2. Randomly choose a student's algorithm for your first try, and narrate as “Java” picks up the “program” to read to you (the computer). Point out that when Java reads code, its compiler translates the code you

type in so the computer can read it in binary. Bonus points if you can get the volunteer to read directions in a robotic voice.

3. Ask your student volunteer reads the directions aloud, following instructions literally. Repeat the instruction out loud as you model the action. Usually students forget to tell you to open the bag of bread, etc. Ham up the errors, and stop “executing the program” when it becomes clear that the algorithm won’t result in a functional PB&J. Point out that this is an error or exception—your program does the same thing when it gets instructions that don’t make sense.
4. Repeat this with 1 or 2 other algorithms, then graciously agree to let the class try again. Make sure to use the phrase “debugging” and have the students use it as well.

1.4.5 Activity Round 2 (10–15 minutes)

1. Give students another 5 minutes to correct their algorithms. If students are on-task and really getting passionate about the job, give them a few extra minutes.
2. Have “Java” select another “program” from your students. As you execute the program again, ask the class what represents **hardware**, **software**, **input**, **output**, **processing**, and the **program**.

1.4.6 Full-class Discussion of Debugging Methods (5 minutes)

1. Ask students to share the different ways they debugged their code/program/algorithm. (Changing program content, switching algorithm order, etc.)
2. Give students with successful algorithms the various PB&Js (failed and otherwise).

1.4.7 Homework Distribution and Exit Tickets (2 minutes)

Distribute homework and have students complete exit tickets.

1.5 Accommodation and Differentiation

Before delivering this lesson, you should check with the classroom teacher to make sure none of the students have a peanut allergy. If a student does have an allergy, find out how severe the allergy is (you can opt to make this student a jelly sandwich, but if the allergy is severe, you should switch the demo to another food item).

In certain classrooms, peanut butter and jelly might not be a familiar food item. In these cases, it is best to do some research first to figure out what snack will be familiar enough to all your students that they can recommend an algorithm for preparing it. Some items (such as Navajo fry bread) might require additional planning. For maximum engagement, try to select a snack that:

- Requires 3 or more ingredients
- Requires assembly or preparation (such as peeling, dicing, etc.)
- Uses ingredients that are jarred or wrapped
- Are generally considered palatable to most students

In ELL classrooms, you should pair students to ease the writing/composition burden of this activity. For advanced students, invite them to write algorithms for other activities, such as getting to school on time.

1.6 Teacher Prior CS Knowledge

- Algorithms are one of the fundamental concepts in computer science. Algorithms are fundamental to solving problems in computer science.

- The lesson plan uses analogy of the student as the compiler. When writing a Java program the following is the sequence of files and translations that are carried out:
 1. Programmer uses an editor to create Java program and saves in .java file
 2. Java compiler (javac program) takes .java file and complies code to bytecode and saves in .class file.
 3. Java virtual machine (java program) runs .class file by interpreting the bytecode for a specific machine based on the operating system and hardware.
- The lesson plan assumes the teacher has prior knowledge of the components of a physical computer and can relate them to a person. If you are not familiar with the components of a computer system:
 - CPU central processing unit – brains of the computer
 - Disk drive/SSD – long term storage
 - RAM random access memory – short term storage
 - Keyboard, mouse, touch screen – input devices
 - Display, sound, vibration – output devices

1.7 Teaching Tips

- Tips for Lecturing: <http://csteachingtips.org/tips-for-lecturing>
- Tips for Introducing Computer Science: <http://csteachingtips.org/tips-for-introducing-computing>
- The lesson plan has the teacher be the “computer” that follows the instructions read aloud by the student. You may be inclined to have a student be the computer. However, students are unpredictable and you may or may not get the outcomes you desire. Even though this activity is meant to be playful, student silliness can get out of hand when building their P&J sandwich. If you do choose to have students be the computer, setup norms like no throwing food and they must clean up after themselves.

Other algorithms you can have students write include tying shoe laces, brushing teeth, opening a locker.

1.8 Misconceptions

Sometimes students think algorithms and computer programs are synonymous. While they are related, they are not synonymous. Humans have been using algorithms to solve problems way before computer ever existed. The invention of the computer created a platform for algorithms created that could be carried out by a machine. So a computer program is a tool used by people to express algorithms that can be executed by a computer.

1.9 Forum discussion

[Lesson 1.02 Algorithms Computational Thinking \(TEALS Discourse account required\)](#)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 1.03 – String & Console Output

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Describe** the history of computer science and Java and why they're used today.
- **Correctly assemble** a complete program with a class header, body, and main method.
- **Correctly use** print, println, and escape sequences.

1.1.2 Assessments — *Students will...*

- **Create** starter Pokémon program

1.1.3 Homework — *Students will...*

- **Read** BJP 1.2
- **Complete** Ch.1 exercises 1-5

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and marker**
- **ASCII Pokémon art**
 - Pokéball: <http://tinyurl.com/pba5x8r>
 - Pikachu: <http://tinyurl.com/oa3g2al>

If you do not have a projector in your classroom, print out pictures of the ASCII art, and place them around the room (or on desks) for students to pass around. Make sure you print pictures out large enough so that students can see the characters that make up the artwork.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction to vocabulary and syntax	10min
Practice questions	15min
Pokémon challenge	10min
Students trade work and debug	5min

1.4 Procedure

In this lesson, you will introduce the parts of a program, then have students create their first “Hello World” style program. Your hook for this class is 2-fold: (1) you should pump up the students to write their very first program ever! (2) Have samples of ASCII art available for them to view, and let them know that they will be creating their own pictures today as well.

1.4.1 Bell-work and Attendance \[5 minutes\]

1.4.2 Introduction to Vocabulary and Syntax \[10 minutes\]

1. Begin your lecture with a quick overview of Java and why we're using it.
 - o Brief history of Java
 - o Key characteristics of Java
2. Lecture on the following talking points. Students should be able to lead you through these points, as you are reviewing the materials from the reading they completed for homework:
 - o Java programs always begin with a **class** header, which follows these rules:
 - o Starts with “public class” (public because anyone can access it)
 - o Uses a capitalized name, and always starts with a letter
 - o Ends with an open curly bracket (think of the curly brackets as a box that holds bits of code together; show students where the close curly bracket goes)
3. Have students volunteer several legit class headers, deliberately make mistakes for students to catch (such as leaving out a bracket, capitalizing incorrectly, or starting class name with a number).

If your students are having trouble generating class headers, guide them through the following examples:

- o `public class MyFile { → correct!`
- o `Public class MyFile { → incorrect, public should be lowercase`
- o `public Class MyFile { → incorrect, class should be lowercase`
- o `public class Myfile { → correct, but not as easy to read file name`
- o `public class WhateverIWant { → correct!`
- o `public class ThisWorks2 { → numbers are OK!`

4. Explain that the “meat” of the program comes from the **methods** (the parts of the program that tell Java to execute a particular action or computation)
 - o You always need a main method, which starts with a method header:

```
public static void main (String[] args) {
```

//

- o Explicitly point out that:

- This is a nonsense list of words for now, but that we'll return to what each part means later on
 - Curly brackets "hold the code together", and so there will always need to be a closed curly bracket at the end of the main method, just like there's a closing curly bracket for the class
5. Ask students to volunteer a short phrase that they would like for their very first program to say (as in "Hello, World!") and use this phrase in your first `println` statement.
- Point out that the statement:
 - Always ends in a semicolon
 - Represents 1 complete order/command
 - Tells Java to print the words within the quotation marks, then go to the next line (`\n`)
 - Have students check the code you've written down on the board. With the class, model how to check code by scanning each line, character by character, having students offer the rules for class and method headers/body, and statements.
 - Erase the "`\n`" from your print statement, and ask students to guess what Java will do with that code (it won't return after outputting the string).
 - Finally, bring students' attention to **escape sequences**, and add some quotation marks to your sample code as an example.

1.4.3 Chapter 1: Introduction to Java Programming Questions (15 minutes)

Have students complete the following questions:

1. Self-Check 1.6: legalIdentifiers
2. Self-Check 1.7: outputSyntax
3. Self-Check 1.8: confounding
4. Self-Check 1.9: Archie
5. Self-Check 1.11: downwardSpiral
6. Self-Check 1.12: DoubleSlash
7. Self-Check 1.13: Sally
8. Self-Check 1.14: TestOfKnowledge

1.4.4 Pokémon Challenge (10 minutes)

On the board or projector, post the following challenge:

Write a program called `Welcome` that outputs the following:

Pikachu welcomes you to the world of Pokémon!

(_) (o^.^) z((")(")

1.4.5 Students Trade Work and Debug (5 minutes)

Have students trade their work and debug each other's programs.

If IDE is available, have students mail you their completed program using the file submission procedure of your choice. Otherwise, have students submit a handwritten form AFTER they have traded their paper with a friend to check and debug.

1.5 Accommodation and Differentiation

If students are struggling with the Pokemon challenge:

- Try pairing up students so they can check each other as they work
- Write the first line of Pikachu code together as a class, modeling the use of escape sequences

If you have students who are speeding through this lesson, you should encourage them to:

- Add additional pictures or text to their Welcome program,
- Help a student that is struggling with the material,
- Create a poster for the classroom with steps (an algorithm!) for checking code for errors (many tips can be found in § 1.3).

1.6 About Pokemon

Throughout the AP CS curriculum, we will gradually be building a larger program around Pokemon, which is familiar to male and female students from all socioeconomic backgrounds, available across the digital divide as both a card game and a video game, and has been translated into 10 different languages (English, Spanish, Portuguese, Dutch, French, German, Italian, Korean, Chinese, and Japanese).

Because the game relies on statistics, modulo operators, and the underlying 32-bit integer that characterizes any given Pokemon, we will be using this theme to introduce students to much of the AP CS curriculum. Students will be entering the AP CS course with varying degrees of math literacy, and framing mathematical challenges in this familiar framework is helpful for avoiding stereotype threat and math anxiety.

In the final project of the course, students will be developing software that uses gameplay ideas similar to those in the Pokemon game.

To learn more about the Pokemon storyline, game rules, underlying formulae, and characters, visit <http://bulbapedia.bulbagarden.net>.

1.7 Teacher Prior CS Knowledge

- The “Hello world!” program is the classic first program taught for many beginner programming classes. It demonstrates the simplest way to get output from the program to the user. The Java “Hello world!” program is chock full of syntax heavy constructs that would not be particularly useful and unduly complicated to a first-time learner to Java. However, knowing these constructs are informative to the teacher: http://www.learnjavaonline.org/en/hello%2c_world%21.

1.8 Teaching Tips

- Tips For Pair Programming: <http://csteachingtips.org/tips-for-pair-programming>
- Tips For Lab Rules: <http://csteachingtips.org/tips-for-lab-rules>
- Explaining “public static void main (String[] args)” would be overwhelming for most beginning Java students. It’s important to let the students know that by the end of the course they will know what the line means but for now all they need to know is to start a Java program, it needs this one line of code.

1.9 Misconceptions

Students learn by making connections to prior knowledge they already know. Unfortunately, this may backfire as in the case of the keyword `class`. When computer scientists use the word `class`, it is automatically assumed that one is referring to class in the context of object oriented programming. However, for a typical high school student, `class` means something totally different: what class am I in now, what homework do I have for math class, or who is the teacher of the class. Even if the student has prior programming knowledge, they may not be familiar with the notion of a class with respect to OOP.

1.10 Video

- CSE 142, *Hello World* (29:24–36:09)
<https://www.youtube.com/watch?v=i2pQHeW5CeY&start=1765>

1.11 Forum discussion

Lesson 1.03 String Console Output

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 1.05 – Static Methods & Method Calls (1/2)

1.1 Overview

1.1.1 Objectives – *Students will be able to...*

- **Use** procedural decomposition to plan complex programs using structure diagrams.
- **Manage** complexity by using method calls

1.1.2 Assessments – *Students will...*

- **Complete** Practice problems

1.1.3 Homework – *Students will...*

- **Read** BJP 1.4
- **Complete** Ch.1 Exercises 11, 12, 14, 16

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and marker**
- **Overly complicated diagram** (<https://tinyurl.com/y67z3cym>)

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction and note-taking	15min
Practice questions	25min
Students trade work, check, and turn in	5min

1.4 Procedure

This class introduces many new concepts in one class period. These concepts will be re-taught in the next class, but you should be aware that your students have a lot of information to absorb in a short amount of time. This lesson will be a good assessment to see if students have been doing their reading and homework, the class should move along smoothly. If students are not completing the readings, you will probably only get through ~50% of the material. If needed, use this opportunity to convince students of the pace and commitment level required for the class.

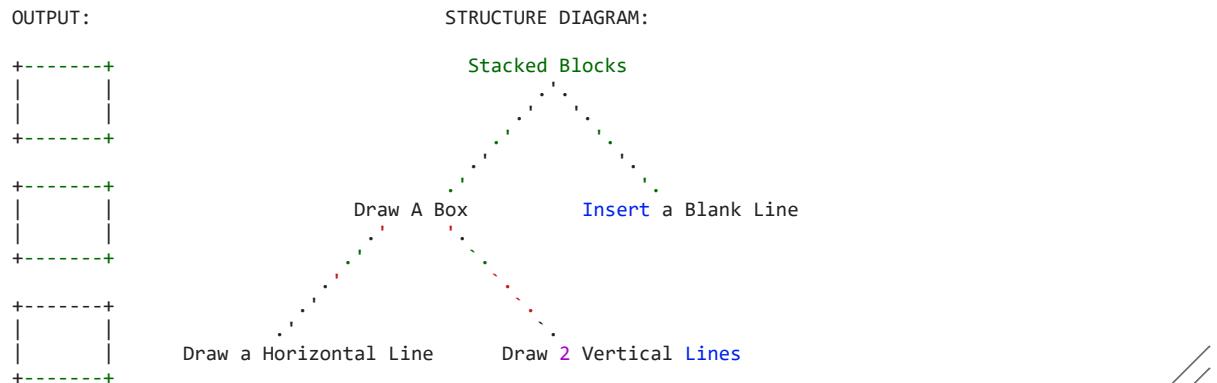
1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Note-Taking [15 minutes]

1. Have the **complicated algorithm** up on the board, or printed out for students to pass around. If you have any confusing furniture assembly manuals, or overly complicated directions, bring those to pass around too.

Start class off with a whole group discussion about why the instructions or diagrams are confusing and ask students what strategies could be used to make them easier to understand. The diagram listed in the materials is an actual slide from the Pentagon, and illustrates how too much complexity can cause all meaning to be lost. Guide the conversation towards decomposition to begin your lecture:

- o **Decomposition**: dividing a problem into smaller, more manageable pieces.
- o **Procedural decomposition**: dividing a whole program into a series of individual steps or actions to program 1 at a time.
- o **Structure diagram**: a way of organizing your approach to building a larger program. Ask students to help you draw a structure diagram for a program with the output shown below:



2. Have students take 3 minutes to write the DrawBoxes program the long way:

```
public class DrawBoxes {  
    public static void main (String[] args) {  
        System.out.println("+-----+");  
        System.out.println("||");  
        ...  
    }  
}
```

3. Point out to students that anything they would cut and paste to save time on creating would make a good unit to turn into a "static method"

- o **Static method** — a block of Java statements that is given its own name (ask students to point 4. to a Java statement)
- o Has the same structure we're familiar with from the main method we already wrote, but we give it a different name than "main".

- A method call interrupts the sequential execution of statements, causing the program to first execute
- the statements in the method before continuing.
 - Once the last statement in the method has executed or a return statement is executed,
 - flow of control is returned to the point immediately following where the method was called.
 - Ask students what simple unit we should build into a static method (have them refer to the structure diagram), and have them suggest a name for the method.
 - Rewrite the method as a class, then show students how to write methods in main that call the new static method. Make sure that students insert the println statements between each method call. It should look something like this:

```
public class DrawBoxes3 {

    public static void drawbox() {
        System.out.println("+-+-+");
        System.out.println("|| |");
        System.out.println("|| |");
        System.out.println("+-+-+");
    }

    public static void main(String [] args) {
        drawBox();
        System.out.println();
        drawBox();
        System.out.println();
        drawBox();
    }
}
```

//

- Call on a student to come to the board and physically trace the flow of control with the marker. Start them off by pointing out that Java always starts with the main method.

If the student seems nervous, encourage the rest of the class to call out directions to the student. Make sure students are drawing the flow of control on their own notes as well.

1.4.3 Practice questions [25 minutes]

- Have students complete the following practice questions:
 - Self-Check 1.22: Tricky
 - Self-Check 1.23: Strange
 - Self-Check 1.26: Confusing
 - Self-Check 1.29: LotsOfErrors-errors

Copy the practice questions to a worksheet and have students complete the practice problems by writing out the answers and using their error-checking algorithm sheets.

Some students will jump right into this activity, but others will need additional assistance from you.

At this point in the school year, we suggest that you insist on structure diagrams with each program. Structure diagrams encourage algorithmic thinking and the creation of efficient solutions; both of which are vital computational thinking skills.

If need be, work on “Tricky” as a whole group, so you can model the correct steps to approaching a problem. If your class decides on an algorithm for “predict the output” type questions, have a student make that algorithm into a poster for the whole class to refer to.

1.5 Accommodation and Differentiation

In English Language Learner (ELL) classrooms, this lesson should be delivered over the course of 2 days. Extra time should be spent drilling static methods, methods that call other methods, and flow of control. Try adapting some of the examples from the book to include students’ native language so they can focus on structuring code

instead of translating language. One easy way to introduce familiar, repetitive content would be to have students output the lyrics to a song with a refrain.

If you have students who are speeding through this lesson, you should encourage them to:

- Complete the remaining
 1. Self-Check 1.24: Strange2
 2. Self-Check 1.25: Strange3
 3. Self-Check 1.27: Confusing2
 4. Self-Check 1.28: Confusing3
- Have the student write a sample test question with output that can be written using method calls. Be sure they include the answer key with the sample question!

1.6 Teacher Prior CS Knowledge

Java has both static and non-static methods. Static methods allow the programmer to call the method without creating an object from the class. Non-static methods covered in the 2nd half of the course requires an object be created from the class before calling the method. Because the *Building Java Programs* textbook introduces functions before objects, methods in early lesson plans are static. For a description of the difference between static and non-static methods see <http://beginnersbook.com/2013/05/static-vs-non-static-methods/>.

1.7 Teaching Tips

- One of the big fundamental concepts of problem solving in computer science is the concept abstraction. This lesson has a number of new syntax constructs for students to create methods and it is important to give students the big picture idea of factoring code and reducing redundancy.
- The practice problems have students tracing code where methods call other methods. Giving students a way of tracing code like using a table to keep track of the method calls. If you are using a black/white board, we recommend crossing out the method call when it completes instead of erasing the name so the student can review the entire flow of control from beginning to end after the exercise is complete. Another technique that can be introduced now that can be used later when the flow of control gets more complex is memory diagrams: <https://www.youtube.com/watch?v=t-TeHodSZs&feature=youtu.be&list=PL0g5FWk3FEqjmrq4ystAvlRyenEF7lUwa>

1.8 Misconceptions

- When declaring a method, students will sometimes incorrectly add a semicolon to the method header, as in `public void foo();`. Students have a misconception that all statements in Java end in a semicolon. They need to know the distinction between statements that do end in semicolon and statements that begin blocks with curly brackets. The addition of the semicolon to the message header could also be students incorrectly pattern matching the method declaration with the method call where there is a semicolon: `foo();` This overgeneralization could lead to semi-colon being incorrectly placed.
- For methods without parameters, students will sometimes omit the parenthesis `()`. To clarify the difference between variables and methods, always use parenthesis when referring to methods. This will reinforce the notion that methods in Java require parenthesis, even for methods with zero parameters.

1.9 Videos

- BJP 1-3, *Programming with Methods*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c1-3
- CSE 142, *Static Methods* (44:12-49:21)
<https://www.youtube.com/watch?v=i2pQHeW5CeY&start=2652>

- CSE 142, *Procedural Decomposition* (20:10–29:35)
<https://www.youtube.com/watch?v=KZYoS7wpMAg&start=1211>
- CSE 142, *Eliminating Redundancy* (29:36–35:49)
<https://www.youtube.com/watch?v=KZYoS7wpMAg&start=1775>

1.10 Forum discussion

Lesson 1.05 Static Methods and Method Calls (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 1.06 – Static Methods & Method Calls (2/2)

1.1 Overview

1.1.1 Objectives – *Students will be able to...*

- **Use** structure diagrams to plan complex programs.
- **Manage** complexity by using method calls.

1.1.2 Assessments – *Students will...*

- **Write** a structured Pikachu program

1.1.3 Homework – *Students will...*

- **Read** BJP 1.5
- **Outline** Ch.1
- **Complete** Programming Project #1 & 3 (must include a structure diagram for each)

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Rosters** for class teams
- **Ball** (preferably a large inflatable ball or kickball)

The teams for today's competition should be your best guess at tiered grouping (these groups will probably change as the year goes on and you learn more about your students). Depending on the size of your class, you should aim for 4 teams or teams of 4 people.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction & challenge	10min
Review/re-teach	5–15min
Pikachu challenge	10–15min
Students trade work, check, and turn in	5min

1.4 Procedure

Today's class re-teaches the many concepts introduced during yesterday's lesson on decomposition, static methods, and methods that call other methods and goes over control flow. Since different students will progress at different rates, we'll begin this lesson with an assessment (in the form of a competition) to determine how much re-teaching you need to do. The competition is really group work in disguise, and will encourage students to teach and help each other while writing the sample program.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Challenge [10 minutes]

- As your hook, grandly announce a class competition between teams and announce the prize for the winning team (this might be TEALS swag, bonus classroom participation points, or additional raffle entries in the year-end TEALS giveaway).
- Announce class teams and rearrange students as needed.
- Announce the 10-minute time limit, set the timer, then reveal the challenge on printed handouts or on the projector with great fanfare. **CHALLENGE:** Write a Java program called StarFigures that generates the following output. You MUST include a structure diagram or your answer will be disqualified. A correct answer will use static methods to show structure and eliminate redundancy in your solution.

```
*****
*****
* *
*
* *
```

```
*****
*****
* *
*
* *
*****
*****
*
```

```
*
*
*
*****
*****
* *
*
* *
```



4. Give students 10 minutes to complete the challenge, and take note of which team finishes first. If students are struggling, you may extend the time, or offer universal helpful tips. The team that has the **correct answer first** wins the prize.

1.4.3 Review/Teach [5-15 minutes]

Review student answers together as a whole group, revisiting concepts taught earlier in the week as mistakes come up. Whenever possible, have students volunteer the correct procedure, approach, or code. Encourage students to take notes during this process so they can review topics over the weekend.

1.4.4 Review/Teach OPTIONAL INSTRUCTION

If students are having trouble understanding the flow of control, you can do this physical activity with them (bonus: this activity can be built on later in the year when discussing return values).

1. Write or project the following code as an example (have students help you with the headers if you are writing):

```
public class SquarePants {  
  
    public static void spongebob() {  
        System.out.println("Well, it might be stupid");  
    }  
  
    public static void patrick() {  
        System.out.print("but it's also");  
    }  
  
    public static void squidward() {  
        System.out.println("dumb.");  
    }  
  
    public static void main (String[] args){  
        spongebob();  
        patrick();  
        squidward();  
    }  
}
```



2. Assign a student to each of the methods: `main`, `spongebob`, `patrick`, and `squidward`, and have them come to the front of the classroom. Have another student act as Java; tell them they are to write the output on the board (or type it if you're using a computer/projector system). From here on out you should address students by their component names (so always call student acting as method `patrick` as Patrick, and so on).
3. Narrate the flow of control as you toss the ball to `main`. Have `main` pass the ball to `spongebob` (and make sure Java "outputs" Well, it might be stupid on the Whiteboard before control passes back to `Main`).
 - o Discuss with the class how Java knows to return to the next call in `main` (the close-bracket), and have the class direct control to `patrick`, then `main`, then `squidward` as Java writes the output on the board.
 - o Pay special attention to the print statement in the `squidward` method, and if kids miss it, make some sort of "error" noise. Bonus: encourage the students to make the error noise for you.

1.4.5 Pikachu Challenge [10-15 minutes]

On the board or projector, have the students finish the class with the Pikachu challenge:

Write a program called PikachuChatter that outputs the following:

Pika pika pika chu pika chu peeeee ka pika chu!
(__/
(o^.^)
z(_(")(")

Pika? Pika pika pika chu peeeeee ka chu!

(__/_)
(o^.^)
z(_(")(")



- You can reuse the code that you wrote earlier this week.
- Your program should use static methods and method calls.
- You should include a comment at the start of the program that explains what the code does (you might want to use this code later in the year when we build a larger Pokéémon program).

1.4.6 Students trade work, check, and turn in [5 minutes]

At the end of class, have students review one another's Pikachu challenge codes before submitting.

1.5 Accommodation and Differentiation

If you have students who are speeding through this lesson, you should encourage them to:

- Complete the remaining problems:
 1. Self-Check 1.24: Strange2
 2. Self-Check 1.25: Strange3
 3. Self-Check 1.27: Confusing2
 4. Self-Check 1.28: Confusing3
- questions.
- Have the student write a sample test question with output that can be written using method calls. Be sure they include the answer key with the sample question!

If you have the good fortune of not needing to re-teach any concepts, you can magnanimously give students extra time to start on the homework programming project \#1. If you are doing a lot of re-teaching during this class, and you feel that students need the emotional reward, you may drop programming project \#1.

1.6 Video

- BJP 1–4, *Drawing Complex Figures with Static Methods*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c1-4

1.7 Forum discussion

[Lesson 1.06 Static Methods and Method Calls \(TEALS Discourse account required\)](#)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 1.07 — Programming Project

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Construct** a program containing method calls and static methods.

1.1.2 Assessments — *Students will...*

- **Submit** a complete, functional program by the end of class

1.1.3 Homework — *Students will...*

- **Check** class notes for completion, adding daily summaries if needed.
 - Students may use the book to supplement their notes if needed.
 - **All students must turn in notes for each day of class** (even if they were absent).

1.2 Materials & Prep

- **Projector and computer**
- **Student self-help system** such as C2B4 ("see two before seeing me") or student pairing

Make sure you are set up to grade student notebooks today while the students work on the project. If possible, you should only collect 3–5 notebooks at a time so students have their notebooks available to reference during programming time.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction & classroom procedure review	10min
Programming project \#2, Chapter 1	15min
Programming project \#5, Chapter 1	15min
Students trade work, check, and turn in	5min

1.4 Procedure

The second week part of this unit will be spent on reinforcing concepts and applying the tools, procedures, and code that were introduced last week. While these classes require little prep before class, you should set up a system that will allow students to help themselves and each other so you aren't running around the computer lab the whole time.

If your computer time requires you to move to another room or to change seating, you should teach and/or review those procedures before introducing the lab material. If you expect students to submit assignments electronically, you should also model and review those procedures before students begin work.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Classroom Procedure Review [10 minutes]

1. Introduce the program assignment, taking a moment to talk strategy with your class.

ASSIGNMENT: Sometimes we write similar letters to different people. For example, you might write to your parents to tell them about your classes and your friends and to ask for money. You might write to a friend about your love life, your classes, and your hobbies, and you might write to your brother about your hobbies and your friends and to ask for money.

Write a program that prints similar letters such as these to three people of your choice. Each letter should have at least one paragraph in common with each of the other letters. Your main program should have three method calls, one for each of the people to whom you are writing.

TIPS: Try to isolate repeated tasks into methods. Include comments in with your code so others can easily understand what the code is supposed to do.

2. Ask your class for suggestions as to how to tackle this programming problem. Students should suggest drawing a structural diagram, building the program one method at a time (iterative development), and following the correction steps on their personal algorithms (debugging).

1.4.3 Programming Project \#2, Chapter 1 [15 minutes]

Get students started on Programming Project \#2 in Chapter 1 of the textbook. Offer students help after they have tried to answer the questions themselves:

1. Have they checked the book for examples?
2. Have they asked a friend (or two) for help?

If students seem to be getting stuck on the same segment of code, offer a hint or tip on the board (silently, without disrupting student flow).

If the entire class is stuck, return to whole group and work through the programming challenge together as a class, having students offer an increasing proportion of the answers as you move along.

1.4.4 Programming Project \#5, Chapter 1 [15 minutes]

Introduce Programming Project \#5 in Chapter 1 of the textbook. If your class finished the first assignment quickly and easily, offer little to no guidance on this project.

1.4.5 Students trade work, check, and turn in [5 minutes]

At the end of class, have students briefly look at each other's projects and review their work before they submit.

1.5 Accommodation and Differentiation

If you have students who are speeding through this lesson, you should encourage them to tackle programming project \#4 in the text book.

1.6 Forum discussion

[Lesson 1.07 Programming Project \(TEALS Discourse account required\)](#)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 1.08 – Finding & Fixing Errors

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Find and fix** errors and style in their returned homework assignments.
- **Correct** their code

1.1.2 Assessments — *Students will...*

- **Re-submit** all homework assignments with corrected answers.

1.1.3 Homework — *Students will...*

- **Study** for the test by:
 - **Reviewing** all the blue pages at the end of Chapter 1
 - **Re-reading** sections as needed
- **Submit** 5 questions for review in class tomorrow using electronic survey

1.2 Materials & Prep

- **Any student homework assignments** that you have not yet returned
- **Student self-help system** (such as C2B4 or student pairing)
- **Electronic survey** for student review requests

When you grade homework assignments, it will be most useful to these lessons if you only mark an answer incorrect or correct. ELL classrooms are the exception to this rule—students will be having a hard enough time just reading the material; you can speed along their processing by correcting one example, then having them look for similar errors with that example.

The homework tonight asks students to submit 5 questions for review. Create an **electronic survey** for students to complete with 6 text fields, one for name, and 5 for questions they have about Ch. 1 content. Set a time-deadline (e.g. 10pm) by which time students must have submitted 5 questions from Ch.1 that they would like to see reviewed in tomorrow's class. If students do not have questions, stipulate that they still have to submit something to receive credit, even if it is only questions they think other students may have.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction and homework distribution	5min
Lecture on style	10min
Student work	25min
Students trade work, check, and submit	10min

1.4 Procedure

Today we continue reinforcing concepts and applying the tools, procedures, and code that were introduced last week. Students will have the opportunity to correct any incorrect homework assignments. If students did not have time to finish the programming projects from yesterday, you may allow them time to work on those projects today.

This is a good day to loosen up the vibe in the classroom a bit. Try playing music softly in the background to encourage students to relax and focus on spotting errors. Try to avoid loud, rhythmic music. Reward your class for good grades/behavior by allowing them to select music from a pre-selected group of Pandora stations (or the like).

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Homework Distribution [5 minutes]

1. Return student homework packets, or have students place their returned homeworks in a pile on their desk.
2. Explain to students that they have the opportunity to get full credit on their homework grades by correcting them now, in class. Ask students for suggestions/ideas on how to make sure they don't miss any errors. (By now students should be used to relying on their error checklist/algorithm.)

1.4.3 Lecture on style [10 minutes]

Add the caveat that they must correct their style to receive that credit. Explain that style is necessary to improve readability and that you lose points for having poor style. - Although it was already touched on previously, tell them that identifiers must be properly capitalized. - Commenting must be properly used throughout. - Lines must be no longer than 100 preferably 80 lines. - There should be no redundant code. - There are links to the full style guide we recommend on the slides.

1.4.4 Student Work [25 minutes]

Have students work individually to correct their homework grades. - Offer time checks for students so they stay on task. - If students have not finished their programming project from yesterday's class, allow them to do so today.

1.4.5 Students trade work, check, and turn in [10 minutes]

At the end of class, have students trade their homework assignments to evaluate each other's corrections before submission.

1.5 Accommodation and Differentiation

If you have students who are speeding through this lesson, you should encourage them to tackle programming project 5 & 7 in the text book.

If you were unable to finish grading student notebooks yesterday, finish them today while students are working. Return notebooks by the end of class so students may use them to study for the exam.

1.6 Forum discussion

Lesson 1.08 Finding and Fixing Errors (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 1.09 — Review

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Identify** weaknesses in their Unit 1 knowledge.

1.1.2 Assessments — *Students will...*

- **Create** a personalized list of review topics to guide tonight's study session.

1.1.3 Homework — *Students will...*

- **Read** Review ch 1
- **Study** for tomorrow's test!

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and marker**
- **Results** from electronic survey of review topics
- **Classroom copies** of the practice test [WS 1.9](#)

Once students have submitted their review requests, assemble those topics into categories and prepare to re-teach the topics as needed.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction and test format orientation	15min
Test review	30min
Check student study lists	5min

1.4 Procedure

Engage the class in the review session by pointing out that your review topics have been hand selected by the class. Explain that you will review test-taking strategies in addition to reviewing subject matter.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Test Format Orientation [15 minutes]

1. Clearly indicate that you expect all students to have a list of review topics to study this evening. Periodically remind students that this list will be checked at the end of class.
2. Begin by explaining the portions of the mock-test. Read aloud the instructions on each page, and explain the strange layout (the test is designed to look like the AP exam, and contains all of the directions on the exam so students will not have to waste time understanding them in May).
3. Solve the sample problems on the test as a whole group, encouraging students to give you the answers whenever possible. Answer any questions that students bring up as you go.

1.4.3 Test Review [30 minutes]

1. Using the results from the electronic survey, address the various review topics, prioritizing questions that popped up the most.
 1. Some questions you may already have addressed while working through the sample test.
 2. Be ready for additional questions to pop up as you go. Save yourself the work and use old homework questions and student-generated test questions as examples to work through.
 3. Jot down notes about which topics you covered in review so you can adjust the exam to reflect the topics your students have learned.
2. Use a combination of group-solving questions on the whiteboard, think-pair-share, and timed-response as review strategies.
3. After you've completed reviewing an idea, remind the class that they should write down that topic if they feel they still have to review it tonight. (Yes, this will be a reminder every few minutes, but it will pay off later when students start creating review lists without prompting later in the year!)

1.4.4 Check student study lists [5 minutes]

Spend the last 5 minutes of class checking each student's review topic list.

1.5 Accommodation and Differentiation

In ELL classes, you may want to change code-writing questions to Parsons Problems. Educational research shows a high correlation between Parsons scores and code writing scores, and a low correlation between code writing and tracing and between Parsons and tracing. (In other words, Parsons Problems accurately assess students' ability to create code.) For more information on Parsons Problems, check out this paper (<https://cseweb.ucsd.edu/classes/fao8/cse599/denny.pdf>).

Even in a non-ELL class, you may want to change some Section II questions to Parsons problems because (1) grading the questions is easier, since logic and syntax errors are easy to discern, and (2) students challenged by language processing are able to more quickly complete the problem.

1.6 Teaching Tips

Tips for Assessment: <http://csteachingtips.org/tips-for-assessing-programming>

1.7 Forum discussion

Lesson 1.09 Unit 1 Test (TEALS Discourse account required)

formatted by *Markdeep 1.093* 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 1.XX — Open-Ended Programming Project (Lesson 1.07 Alternative)

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Ideate and construct** a program containing method calls and static methods.

1.1.2 Assessments — *Students will...*

- **Submit** a complete, functional program by the end of class

1.1.3 Homework — *Students will...*

- **Check** class notes for completion, adding daily summaries if needed.
 - Students may use the book to supplement their notes if needed.
 - **All students must turn in notes for each day of class** (even if they were absent).
-

1.1.3.1 Emphasize with students...

1.1.3.2 Big Ideas - Personal design interests require the evaluation and refinement of skills

Computers are great at doing repetitive tasks. In computational thinking, we use the words “decomposition”, “pattern recognition”, “abstraction”, and “algorithm design”. Think of the problem you are given. Break down the problem into manageable chunks. Are there any parts that show patterns, or similarities? Let’s take advantage of the computer to help us do “similar” looking tasks.

In daily life, human beings enjoy repetition and patterns in many areas of life and nature. For the written word, repetition is useful for the learning of songs or poetry. It's a literary technique. Repetition enhances significance and meaning.

Consider the [national anthem of USA](#) Did you know that there is an English version, and a [Spanish version](#)?

Imagine that you are creating a national anthem medley for your class to perform at a special event. Your challenge is to include a combination of language versions. How can you use static methods to design this musical piece?

1.2 Materials & Prep

- **Projector and computer**
- **Student self-help system** such as C2B4 ("see two before seeing me") or student pairing

Make sure you are set up to grade student notebooks today while the students work on the project. If possible, you should only collect 3–5 notebooks at a time so students have their notebooks available to reference during programming time.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction & classroom procedure review	10min
Programming project	30min
Students trade work, check, and turn in	5min

1.4 Procedure

The second week part of this unit will be spent on reinforcing concepts and applying the tools, procedures, and code that were introduced last week. While these classes require little prep before class, you should set up a system that will allow students to help themselves and each other so you aren't running around the computer lab the whole time.

If your computer time requires you to move to another room or to change seating, you should teach and/or review those procedures before introducing the lab material. If you expect students to submit assignments electronically, you should also model and review those procedures before students begin work.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Classroom Procedure Review [10 minutes]

1. Introduce the program assignment, taking a moment to talk strategy with your class.

ASSIGNMENT: Sometimes we write similar letters to different people. For example, you might write to your parents to tell them about your classes and your friends and to ask for money. You might write to a friend about your love life, your classes, and your hobbies, and you might write to your brother about your hobbies and your friends and to ask for money.

Write a program that prints similar letters such as these to three people of your choice. Each letter should have at least one paragraph in common with each of the other letters. Your main program should have three method calls, one for each of the people to whom you are writing.

TIPS: Try to isolate repeated tasks into methods. Include comments in with your code so others can easily understand what the code is supposed to do.

2. Ask your class for suggestions as to how to tackle this programming problem. Students should suggest drawing a structural diagram, building the program one method at a time (iterative development), and

following the correction steps on their personal algorithms (debugging).

1.4.3 Programming Project [30 minutes]

Reference the exercise above, where you write (ie, output, or print) similar letters to different people. The main program has common parts. In the main method, make calls to static methods that do other variations.

Examples of text-based output that have this characteristic:

- A standardized form, or letter
- Lyrics of a song or rhyme that has:
 - Lines that are repeated at each verse, e.g. "There Was an Old Lady" (Reference text book Chapter 1, Project #2)
 - New lines are added onto existing old lines, e.g. "Twelve Days of Christmas", "The House That Jack Built". (Reference text book Chapter 1, Project #3, #5.)
- Lyrics of a song that includes words (or sections) in different languages
- A cheer or chant for a school team
- A medley of different songs (verse and chorus) put together

Static methods can simplify output of the above task. For example, you can use methods for each verse, and for repeated text.

What other types of output have repeated patterns? What are other types of printed output that can benefit from static methods? Choose one to implement.

1.4.4 Students trade work, check, evaluate and turn in [5 minutes]

At the end of class, have students briefly look at each other's projects and review their work before they submit.

Evaluation Question: How many lines of code did this program "save" by calling static methods, instead of calling print statements to output every single line of text?

1.5 Accommodation and Differentiation

If you have students who are speeding through this lesson, you should encourage them to tackle programming project \#4 in the text book.

1.6 Forum discussion

[Lesson 1.XX Programming Project \(TEALS Discourse account required\)](#)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 2.00 — Test Review & Reteach

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Re-learn or strengthen** content knowledge and skills from Unit 1.

1.1.2 Assessments — *Students will...*

- **Re-submit** test answers with updated corrections for partial or full credit
 - Credit depends on instructor preference

1.1.3 Homework — *Students will...*

- **Read** BJP 2.1 except for “Mixing Types and Casting”
- **Correct** any incorrect test answers by re-answering on a separate sheet of paper
 - To get back credit, they must justify their new answers
 - Staple new answer sheet to old test and turn in tomorrow

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Corrected student tests**
- **Student grades** (posted online, emailed to students, or handed back on paper in class)
- **Digital copy of test questions** for projector

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Class discussion (if needed)	10min
Test review and reteach	30min
Check student notes and return tests	5min

1.4 Procedure

Return student grades before class begins or while students are completing the bellwork.

Do not return students' tests before the review session, since you want to motivate students to pay attention to the entire review, taking supplemental notes the entire time.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Class Discussion (if needed) [10 minutes]

1. If grades are low, invite the class to a discussion of what can be improved. Begin with student complaints and suggestions to build student buy-in. Ask students:
 - How they felt they were going to do before the test
 - What surprised them once they were taking the test
 - What they felt worked in the first unit (lessons, review strategies, assignments)
 - What do they think they want to change for the second unit
2. Once you feel that a dialogue has been established, validate students' feelings, then challenge them (e.g. AP courses are stressful, but this is good practice for college, where the pace is faster and professors don't give personalized instruction). In a non-judgmental, supportive tone, remind students that to be successful in the course:
 - Reading is mandatory
 - Homework is mandatory (And valuable! You will never assign "busy" work.)
 - To better manage their time, students should plan for 1 hour of homework a weeknight, with up to 2 hours of homework each weekend. If this seems impossible, they should meet with you or their guidance counselor to assess whether they can fit in an AP class at this time.
 - It is VERY important to keep your tone sympathetic at this point—an overworked, overstressed, underperforming student will slow your entire class down, and color that student against CS for the future!

1.4.3 Test Review and Reteach [30 minutes]

1. Walk the students through each question on the test, glossing over questions that everyone answered correctly.
 - a. You can ask for students to volunteer answers, or call on students randomly. Make sure that students explain their logic when they answer. If a student gives an incorrect answer, the explanation will tell you what you need to re-teach or clarify.

- b. Do not skip questions that everyone answered correctly, but do not spend more than the time it takes to read the question, and congratulate students' correct answers.
- 2. Project a copy of each question as you review—this will help students recall the question/process the information.
- 3. Make sure that students are taking notes during the re-teach, reminding students that for homework, they will have an opportunity to win back some of the points on their exam.
- 4. For Section II questions, select a sample of student work (with any identifying information obscured), and work through the answer together as a class.

1.4.4 Check student notes and return tests [5 minutes]

At the end of class, check student notes, and return the tests in hard copy form if applicable.

1.5 Accommodation and Differentiation

The curriculum does not officially cover the char type since it is not included in the AP subset. However, if your class is progressing quickly, feel free to introduce char into all future examples, worksheets, and tests.

In ELL classrooms, you should give more examples for each type, and spend more time drilling during the introduction and note-taking segments.

1.6 Forum discussion

[Lesson 2.00 Test Review and Reteach \(TEALS Discourse account required\)](#)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 2.01 – Basic Data Concepts

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Identify and categorize** data types
- **Identify** operators and operands.
- **Correctly apply** rules of precedence

1.1.2 Assessments — *Students will...*

- **Write** code that yields a given answer, using rules of precedence
- **Create** expressions and predict output using operator/operand expression sets

1.1.3 Homework — *Students will...*

- **Complete** self-check questions 1-3 (4th edition: 1, 3, 4)
- **Read** BJP 2.2 up to "String Concatenation"

1.2 Materials & Prep

- **Projector and computer**
- **White paper and markers**
- **Classroom sets** of operator/operand expression cards created from [WS 2.1](#)
- **Pair or small group** student assignments

Operator/Operand Expression sets can be printed and cut from regular printer paper, or you can write them out on construction paper, creating color-coded sets (recommended to prevent cheating and reinforce memory cues).

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction to data types	10min
Think-pair-share activity	5min
Introduction to operators and precedence	10min
Evaluating Expressions activity	20min
Check student study lists	5min

1.4 Procedure

Hook your class today by explaining that they're going to be able to create a calculator by the end of this week.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction to Data Types [10 minutes]

Begin with a brief lecture about data types.

- To write a more complicated program like a calculator, we need to familiarize ourselves with the different types of data that Java can work with.
- **Type** (or data type): a name for a category of data values that are all related
 - **Primitive data types:** Store the actual value in the variable
 - **Type int** describes all whole numbers, or integers (have students name some examples)
 - -int variables take up 4 Bytes of space in memory (see chart below)
 - -Remember to talk about data overflow and what happens if you put too large a value into an int
 - **Type double** describes all numbers with decimal points (have students give some examples)
 - double variables take up 8 Bytes of space in memory (see chart below)
 - You can remember if something is a double because there are numbers on both sides of a decimal point (like 2 numbers, double numbers)
 - **Type boolean** describes logical values—this means true or false. There are no other values in type boolean.

1.5 Pacing Guide

Data Type	Total Size	Range of Values
int	4 Bytes	-2,147,483,648 to 2,147,483, 647
double	8 Bytes	approximately 15 significant
		decimal digits
boolean	1 bit	true or false

- **Reference data types:** store the location in memory of the object the code is referring to
 - -A String variable would be an example of a reference data type
- An **expression** is a simple value, or a set of operations (an equation) that produces a value.
 - One simple example of an expression is value, like 3.14 or 439.
 - Another example of an expression is $2 + 5.9$, because it is an operation that produces a value. (Ask students to point out the int, double, and expression in this example.)
- In the expression $2 + 5.9$, the plus sign is called an **operator** because the symbol indicates an operation to be performed on one or more values.
 - We refer to the values as **operands**—both int and double are operands.

1.5.1 Think-Pair-Share Activity [5 minutes]

1. While students are finishing writing down definitions in their notes, write an assortment of data type examples on the board.
2. Have students categorize all of the primitive types on the board during a Think-Pair-Share exercise. Remind students to do scratch work in their notebooks, since it will count towards their classwork grade (this encourages everyone to work during the “think” stage of the activity).
3. Bring the class back to whole group, and call on students to share a category for each data type.

1.5.2 Introduction to Operators and Precedence [10 minutes]

1. Do a quick review of arithmetic operators. Students should be able to volunteer most of these, but you may have to spend some review on mod, especially if your class is not on grade level for mathematics.
2. Ask students for the operators that represent addition and subtraction.
3. Introduce the special symbols we use for the operators multiplication and division.
 - Division has slightly different rules if you’re working in type int:
 - $12 / 5$ evaluates to 2, because even though the calculator shows us 2.4, int doesn’t let us have a decimal point (what type does?)
 - It’s very important to remember that int always drops the part after the decimal point. So even if you evaluated $39 / 10$, your answer would be 3, not 4.
4. Introduce the **mod % operator**, and have students work through a few examples with you to practice.
 - In elementary school we called it a “remainder”
 - $1079 \% 34$ evaluates to 25, because you get 31 R 25 (34 goes into 1079 34 times, with 25 left over)

- If you try to get the answer with your calculator, you won't get 25—you'll need to do long division to get the right answer (or, you can get Java to do it!)
5. If we don't use parentheses in our expressions, Java uses **precedence** to decide which operations go first (students will probably mention PEMDAS), and evaluates left-to-right:
- $13 * 2 + 239 / 10 \% 5 - 2 * 2$
 - Start left to right, $13 * 2$ evaluates to 26
 - $239 / 10$ evaluates to 23 (have students do this one to see if they catch the int)
 - Still moving left-to-right, now $23 * 5$ evaluates to 3, and $2 * 2$ evaluates to 4
 - $26 + 3 - 4$ evaluates to 25
6. Quick way to round doubles. We just showed that when doing division we drop the decimals so here is a quick way to round doubles:
- for positive numbers:
 - $(int)(x + 0.5)$
 - for negative numbers:
 - $(int)(x - 0.5)$

1.5.3 Evaluating Expressions Activity [20 minutes]

1. Depending on your class size, have students form pairs or small groups
2. Give each pair or small group a Ziploc bag with a set of operand/operator cards.
3. Students should write out the expressions they create, along with the value they evaluate to, in their notebooks.
4. Once students have finished a set, have them repack the set and trade with another group (or trade in their set with you).
5. Encourage groups to check each others' answers and help each other if they get stuck.

1.5.4 Check Student Study Lists [5 minutes]

At the end of class, go over student notebooks.

1.5.5 College Board Topic Question

After this lesson, students will be able to answer questions from the College Board Unit 1 Topic Questions 1.2: Variables and Data Types

1.6 Accommodation and Differentiation

The curriculum does not officially cover the char type since it is not included in the AP subset. However, if your class is progressing quickly, feel free into introduce char into all future examples, worksheets, and tests.

In ELL classrooms, you should give more examples for each type, and spend more time drilling during the introduction and note-taking segments.

1.7 Teacher Prior CS Knowledge

- The AP CS A exam covers a subset of the Java primitive data types. For a more though understanding of the Java data types (byte, short, int, long, float, double, char, and boolean) see http://www.learnjavaonline.org/en/hello%2c_world%21.

- String is not a primitive data type in Java but is a class. Strings in many behaves like a primitive data type, for example you can add two Strings together with the + sign. This is a source of confusion for many beginner Java programmers as the language is inconsistent with its treatment of String.

1.8 Video

- BJP 2–1, *Expressions*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c2-1
- CSE 142, *Basic Data Concepts* (40:42–49:59)
<https://www.youtube.com/watch?v=KZYoS7wpMAg&start=2442>

1.9 Forum discussion

Lesson 2.01 Basic Data Concepts (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 2.02 – Declaring & Assigning Variables

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Identify, declare, and assign** variables.

1.1.2 Assessments — *Students will...*

- **Write** a program that converts temperature from Fahrenheit to Celsius.

1.1.3 Homework — *Students will...*

- **Read** the rest of BJP 2.2
- **Complete** self-check questions 5, 6, 9, 12-15 (4th edition: 6, 7, 10, 14-17)

1.2 Materials & Prep

- **Projector and computer**
- **White paper and markers**
- **Classroom copies** of [WS 2.2](#)
- **Pair or small group** student assignments
- **Sample online temperature converter** (<http://www.onlineconversion.com/temperature.htm>)

Since most of today's lesson follows WS 2.2, you should have read through the worksheet. You may prefer to delete the notes from the worksheet (so it is only a sheet of exercises) if you are working on developing note-taking skills in your classroom. We recommend leaving these sections in for English Language Learner (ELL) classrooms, so your students can focus on syntax rules instead of translating what they are hearing to vocabulary they need to then write in their notebooks.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction and worksheet exercises	25min
Practice exercises	20min
Turn in worksheets, wrap up	5min

1.4 Procedure

Since much of this class involves learning syntax, there will be a lot of practicing during the class. Try to spice up the lesson by allowing students to work in pairs, or playing soft music in the background to put students in the right headspace to settle down for work.

Hook your class today by asking which of them are taking or have taken physics or chemistry. Ask students about working with Fahrenheit and Celsius temperatures—do they have to convert temperatures in class? Which measurement are they more familiar with? Which do they use more often? Show students the online calculator and ask if they ever use such online tools, and tell students that they’re going to learn how this program is built today.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Worksheet Exercises [25 minutes]

1.4.2.1 Emphasize with students...

1.4.2.2 Content - Debugging Tools

As you continue to program in IDE you will notice some of the debugging tools that are available to you. One of these tools is the syntax coloring whereby syntax errors will be highlighted or underlined a specific color. This draws the programmer’s attention to the error and allows for quicker fixes.

Even though these tools exist, it is still important for you to get good at identifying syntax and other errors. This will speed up the development process and will result in fewer errors in the final program.

1. Using WS 2.2, walk students through the proper way to declare a variable.
 - Be sure to spot-check for understanding by having students give you the definitions of **type**, **syntax**, **declaration**, and **variable** (all bolded in the text).
 - Encourage students to use their notes if needed.
2. Guide students through the syntax rules for variable declarations by working through the first few examples of Exercise 1 in pairs.
3. Give students a few minutes to complete Exercise 1 on their own; encourage students to tackle Exercise 2 as well, then check all answers together as a whole group.

- Using the figure on Exercise 3 of WS 2.2, walk students through the proper syntax to assign a variable.
 - Spot-check for understanding by asking students to define the italicized words.
 - Ask students for a few sample answers, correct them if needed, then give students a few minutes to complete Exercise 3 in pairs.
- As a whole group, walk students through Exercise 4 and 5. Complete 5a together as a group, then let students work on 5b in pairs.

At this point, your class may be raring to get started on the rest of the assignment without your help. If they are, great! Post the practice questions on the board so they can continue to that assignment once they have completed the worksheet. If your class wants you to walk them through string concatenation, go through the examples as above.

1.4.3 Practice Exercises [20 minutes]

- Have students complete the following practice self-check questions:
 - Self-Check 2.8: studentVariables
 - Self-Check 2.13: valuesOfABC
- Have students complete Exercise 2.1: displacement.
- Students should work on their own, but if the exercise is too challenging, you might opt to have students collaborate on answers. Be sure to remind students that each student should turn in their own set of work.

1.4.4 Students turn in worksheets, wrap up [5 minutes]

At the end of class, collect WS 2.2 and practice problems submissions.

1.5 Accommodation and Differentiation

If you have students who are speeding through this lesson, you should encourage them to:

- Complete Self-Check 2.18: timesOperator.
- Challenge them to build their own program that converts Fahrenheit to Celsius (this version won't take user input—yet!)
- Have the student create a classroom poster diagramming the parts of variable declaration & assignment.

If your class is struggling with learning syntax, you can split the lesson into 2 lessons, and/or take off some of the homework questions. If splitting the lesson in two, we recommend stopping today's lesson before string concatenation.

1.6 Common Mistakes

Variables

common

mistakes:

<http://interactivepython.org/runestone/static/javareview/variablebasics/commonmistakes.html>

1.7 Misconceptions

- Students will draw on their math knowledge when learning variables. This leads to confusion on the differences in a programming language.
- The equal sign = is assignment in Java, not equality. When reading out code, explicitly saying “assignment” will help reinforce the concept that = is not equality: $x = 1$ is read as “x is assigned the value 1”.
- A variable is a container for value that can change, it does not denote a fixed value. From WS2.2, consider the following statements:

```
int age = 17;  
age = age + 1;
```



Students may view the second line of code as a math equation and miss the point that age is a variable and contains a value. The $age = age + 1$ changes the value of age. When learning Java there is the code which is static, the running of the code, and the state of the variable during runtime which is fundamentally different from an equation in math that can be substituted and manipulated to solve for some unknown.

- Students read $y = x + 2$; and think the *equation* is stored in y, not a *value*.

```
x = 1;  
y = x + 2;  
x = 3;  
System.out.println(y); // what is displayed for y?
```



For students with the misconception that the *equation* is stored, they will incorrectly compute 6 as what is displayed for y.

Java is not a spreadsheet that stores the equation. Explicitly teaching that x and y are independent variables in a programming language and hold values is an important distinction. This differs in math where $y = x + 2$ is a relationship. Showing the state of the variables can be achieved using the whiteboard, the debugger by stepping through one line of code at a time, and examining variables, or using a visualization tool like http://cscircles.cemc.uwaterloo.ca/java_visualize.

1.8 Video

- BJP 2–2, *Variables and Assignment*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c2-2
- CSE 142, *Variables* (5:12–12:48)
<https://www.youtube.com/watch?v=oeUm1RGkWw&start=310>
- CS Homework Bytes, *Variable and Assignment, with Elizabeth*
https://www.youtube.com/watch?v=fPqGiexXi_Y

1.9 Forum discussion

Lesson 2.02 Declaring and Assigning Variables (TEALS Discourse account required)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 2.03 – String Concatenation & Increment Decrement Operators

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Apply** the rules of string concatenation.
- **Correctly interpret** incrementing and decrementing statements.

1.1.2 Assessments — *Students will...*

- **Evaluate** statements and predict output during a game of Grudgeball

1.1.3 Homework — *Students will...*

- **Read** remainder of BJP 2.2
- **Complete** self-check question 4 (4th edition: 5)

1.2 Materials & Prep

- **Projector and computer** (*optional*)
- **White paper and markers**
- **Rules** for Grudgeball (see website for details: <http://toengagethemall.blogspot.com/2013/02/grudgeball-review-game-where-kids-attack.html>)
- **Team assignments** that divide your class into 5 or 6 teams
- **Nerf hoop & ball** (or wastepaper and trash can)
- **Taped 2- and 3-point lines**

Take the time to familiarize yourself with the rules of Grudgeball, and test out your 2 and 3 point lines before class begins (you may need to readjust them). If you can get permission from your school to leave tape on the floor, it is helpful to have those lines down for the rest of the year. In future classes, if your students are having a hard time settling down during a review session, or can't stand a worksheet, you can always convert the worksheet or review session into a quick game of Grudgeball.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction and note-taking	15min
Grudgeball	35min

1.4 Procedure

Because today's lesson combines several related concepts with no main theme, the drilling/activity portion of the class will serve to tie the lesson together in the form of a class competition. If space and whiteboard setup allow, set up the Grudgeball "court" and scoreboard before class begins so it hooks the students as they walk in. Before you begin lecture, announce to students that they should pay close attention, since the lecture content will be tested during the game.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Vocabulary and History of Bugs [10 minutes]

1. Begin with a lecture about the topics to be covered.

- o **String concatenation:** lets you put together several strings into one string, the way we combine numbers in an expression!
- o When you see a + between strings (look for those quotation marks!), that means that you "add" the strings together:

```
System.out.println("Spongebob thinks " + "the best time to wear" + " a sweater" + "is all the time!"); //
```

Carefully write out (or point out, if using a Powerpoint) the spaces within the Strings. Write out how the string concatenates, asking students to predict the next portion of the string combo.

- o Offer up a second example, asking the students to predict how the expression will evaluate, discussing the placement of quotation marks first:

```
System.out.println("Here we combine " + 1 + " integer" + "with the strings!"); //
```

- o Give the students a little extra guided practice by walking them through this example:

```
42 + " is the answer to " + "everything!" + 1 + 1 //
```

- Stepwise, show students how this evaluates to "42 is the answer to everything!11"
- Ask students if they can come up with a way to make the answer evaluate to "42 is the answer to everything!2"

2. Switch gears and tell students the second useful tool you're going to teach them today is how to **increase (or decrease) a variable** without writing a whole separate equation.

- o Briefly explain that in Java, the equations below mean "add 7 to the current value of x," "divide the current value of y by 3," instead of the traditional mathematical use of "equals." Immediately show students how to write the abbreviated versions of these statements:

- x = x + 7; OR x += 7;
- y = y / 3; OR y /=3;

- Once students have wrapped their heads around this non-algebraic reading (and given you some correct examples), explain the special case of incrementing or decrementing by 1:
 - x++; OR ++x;
 - x--; OR --x;

1.4.3 Grudgeball [35 minutes] [Optional]

If you feel like your class understands increment and string concatenation, consider skipping this game and focusing on on-the-board examples (you can use the questions from Grudgeball below) or moving on to 2.4.

- Divide students into their assigned teams.
- Review the rules for Grudgeball, and have the students repeat the rules back to you.
- Using the problems listed below (and any you may add, depending on your class' needs), play Grudgeball until a team wins, or until the class period ends.
 - If a class gets the answer wrong, BRIEFLY pause the game to have students offer corrections before moving to the next team's question.
 - If correction seems to be dragging on, jump in and quickly re-teach using the incorrect answer as your example. It is important to keep the pace going to maintain student interest in the game!

Grudgeball problems & answers have been grouped assuming that you have 6 teams. If you have fewer teams, each "round" will be shifted accordingly, so you may have rounds where different teams are practicing different concepts. Judge each team's knowledge gaps, and adjust which questions you ask each group accordingly.

1.4.3.1 Grudgeball PROBLEMS AND ANSWERS

1.4.3.1.1 What do these evaluate to?

- "Patrick" + " why" + "are you" + "here?" → Patrick whyare youhere?
- 2 + "words: " + "Na. Chos." → 2words: Na. Chos.
- "Friendship " + 1 + "\$" + " magic!" → Friendship 1\$ magic!
- "Watch out" + " for " + "\"\" + "" + "escape sequences!" → Watch out for ""escape sequences!"
- "Pikachu, pika pika" + "peeeeeekaa" + " ch" + 0 + 0 +"!" → Pikachu, pika pikapeeeeka ch00!
- "PEMDAS" + "doesn't " + (2 + 3) * 4 + "matter " + "right?" + 1 → PEMDASdoesn't 20matter right?1

1.4.3.1.2 Write a statement that:

- Increases the current value of x by 150. → $x = x + 150$; or $x += 150$;
- Decreases the current value of y by 9. → $y = y - 9$; or $y -= 9$;
- Multiplies the current value of z by 5. → $z = z * 5$; or $z *= 5$;
- Divides the current value of q by 14. → $q = q / 14$; or $q /= 14$;
- Increments x by 1. → $x++$; $++x$; $x = x + 1$; or $x += 1$;
- Decrements x by 1. → $x--$; $--x$; $x = x - 1$; or $x -= 1$;

1.4.3.1.3 Predict the output:

13)

```
int x = 1;
x += 3;
System.out.println("The value of x is " + x);
```

→ Output: The value of x is 4

14) 1 + 1 + 1 + "1" + 1 + 1 + 1 → Output: 31111

15)

```
int y = 2;
y /= 2;
System.out.println("1 + " + y + "is how much again?");
```

→ Output: 1 + 1 is how much again?

16) 110 - 10 + "flip it " + 0 + 0 + 1 → Output: 100flip it 001

17) "100 - 10" + "flip it " + 0 + "0 + 1" → Output: 100 - 10flip it 00+1

18)

```
int number = 5;
number++;
System.out.println("My new value" + "is the " + "number " + number);
```

→ Output: My new valueis the number 6

1.4.4 College Board Topic Question

After this lesson, students will be able to answer most of the questions from the College Board Unit 1 Topic Questions 1.3: Expressions and Assignment Statements and 1.4: Compound Assignment Operators.

Do not assign Boolean operator or two dimension array questions from 1.3 as these topics are covered in future lessons.

1.5 Accommodation and Differentiation

If your class is struggling with learning string concatenation and/or incrementing decrementing, the best strategy here is to repeat, repeat, repeat. Add more simple problems before you advance to the mixed type concatenation, and work through more of the problems as a whole group.

In ELL classrooms, you should read each question aloud in addition to showing it on the board or projector.

1.6 Common Mistakes

Common mistakes with strings: <http://interactivepython.org/runestone/static/javareview/strings/smistakes.html>

1.7 Videos

- CSE 142 (12:48–18:29)
<https://www.youtube.com/watch?v=oeUm1RFGkWw&start=769>

- CS Homework Bytes, *Mathematical Operators and Precedence, with Vinnie*
<https://www.youtube.com/watch?v=RTmRwEy-yFA>

1.8 Forum discussion

Lesson 2.03 String Concatenation & Increment Decrement Operators (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 2.06 — nested for Loops

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Trace** nested loops to predict program behavior.
- **Construct** loops to execute simple tasks.

1.1.2 Assessments — *Students will...*

- **Trace and construct** nested loops in practice problems.

1.1.3 Homework — *Students will...*

- **Read** BJP 2.3 “Nested for Loops”
- **Complete** self-check questions 26, 27 (4th edition 29, 30) and exercise 4

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and marker**

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction to nested for loops	15min
Practice Activity	30min
Loop Challenge	5min

1.4 Procedure

Today your hook is another wager, only this time the students have to create the code to get the reward (TEALS swag, tickets to the raffle, bonus points, etc.). If you plan to use a badge system, this project is a good one to start with (a ‘loop’ badge, perhaps?).

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction to Nested for Loops [15 minutes]

1. Remind students of yesterday’s bet, then tell your students that you’ll award them [prize] if they can rewrite the program from yesterday in 11 lines. The catch is that student can’t use a loop that executes more than 10 times! Explain that you’re going to give them a new programming tool today, and then they’ll have some time at the end of class to see if they can meet your challenge for the points.
2. Review the concept of “**control structure**,” and have students explain to you that the loop controls the statements in the loop body. Ask students to walk you through the example on the board, narrating what Java is doing as you advance your hand along the loop:

```
for (int i = 1; i <= 3; i++) {  
    System.out.println("Calacas y calaveras!");  
}
```



You might consider deliberately leaving in an error or two as you write the code on the board—students LOVE to catch you in an error! In general, leaving in errors is a good way to condition error-checking behavior in your students without them feeling forced. If students don’t catch the error before you’re ready to move on, point to it and ask students if the code is correct. Occasionally point to correct code and ask students to explain why it is a proper coding choice, or what other options would be.

If your class does not celebrate Día de los Muertos, change the string above to “Trick or Treat,” “All Saints Day,” “Kol Sana Wenta Tayeb” or whatever is most relevant.

If your class needs more physical engagement, have a volunteer come to the board to be console output, and point to them, directing them to write output on the board every time the loop is executed. You can also have another student walk through the flow of the loop in your place.

3. As you insert another loop to create a nested loop, explain that the great thing about control structures is that they can control other control structures!

- (Engagement option: if your class is familiar with Xzibit/Pimp my Ride, this is a great opportunity for a yo-dawg meme, but at this point, this is probably only a reference that college-age and above will get.)

```
for (int i = 0; i < 3; i++) {  
    for (int j = 1; j <= 3; j++) {  
        System.out.println("Calacas y calaveras!");  
    }  
}
```



- Point out that we always use a different control variable (j instead of i) so that Java knows we’re writing a new loop.
- As you write the inner loop, ask students how many times it executes, and briefly discuss the difference between initializing at 0 and 1, and how that relates to < or <= in the test.
- Ask students how many lines Java will output (9), and walk through the loop showing flow of control and directing a student to produce the output.

1.4.3 Practice Activity [30 minutes]

Depending on the mood and frustration levels in the class, you may choose to have students work in pairs.

- If students are really having a rough time, work through the first practice question together as a whole group.
- Put soft, soothing (but upbeat) music on in the background to ease tension!
- Have students complete the following self-check questions:
 - a. Self-Check 2.31: starExclamation1
 - b. Self-Check 2.32: starExclamation2
 - c. Self-Check 2.33: starExclamation3
- Have students complete the Exercise 2.5: starTriangle.
 - a. You should emphasize the importance of constructing a structure diagram before they start coding.
 - b. Although we haven't talked about pseudocode yet, suggest that students write out in English (or whatever their preferred language is) the steps that they will need to do to solve the problem. Encourage students to work on this in pairs if needed.
- If more 25% or more of the class is struggling, return to whole group with the stipulation that students who get it may continue working independently.

1.4.4 Loop Challenge [30 minutes]

1. On the board or the projector, bring up the challenge you introduced in the beginning of class.
 - Students may begin on the challenge once they have finished their practice exercises.
 - In order to encourage all students to try the challenge, allow slower students to submit their challenge answer electronically by the end of the day. (This compromise gives them time to work on the code at lunch or after school, but dissuades students from directly copying others' answers.)

1.4.4.1 LOOP CHALLENGE

Write a program that outputs the first 1,000 integers in 11 lines of code. You may not use a loop that executes more than 10 times.

```
public class Count1000 {  
    public static void main (String[] args) {  
        for (int i = 0; i < 10; i++) {  
            for (int j = 0; j < 10; j++) {  
                for (int q = 0; q < 10; q++) {  
                    System.out.println((i*100)+(j*10)+q+1);  
                }  
            }  
        }  
    }  
}
```



1.5 College Board Topic Questions

After this lesson, students will be able to answer questions from the College Board Unit 4 Topic Questions 4.2 For Loops and 4.4 Nested iteration.

1.6 Accommodation and Differentiation

If you have students who are speeding through this lesson, you should encourage them to:

- Complete Exercise 2.3: fibonacci and Exercise 2.6: numberTriangle.
- Develop an algorithm for tackling complex coding problems. (What generalizable steps did they take to correctly build the nested loop programs?)

Have the student turn these ideas into a mnemonic, poster, or checklist to share with the class.

1.7 Misconceptions

- Determining the purpose of the two variables counters in nested loops is confusing to students. Most introductions to nested loops use "i" and "j" as the loop variables. Starting with "i" is a carryover from the Fortran programming language where variables starting with the letters I to N were integers and loop variables are integers. However, Java does not have this restriction, integer variables can start with any letter.

In order to help students grasp the two loop variables, use loop variables in context where students may be more familiar with: row/column, x/y. This affords using a graphical representation that students can plot while tracing through the nested loops.

- Confusion of the order of execution of the 3 parts of the for loop from a single for loop gets compounded with nested for loops. The order of execution for the nested loop is:
 - outer loop initialize variable
 - outer loop test condition
 - inner loop initialize variable
 - inner loop test condition
 - body of inner loop
 - inner loop update variable
 - repeat inner loop
 - outer loop update variable
 - repeat outer loop
- Attempting to use the inner loop variable in the outside loop block. It is not obvious to beginners that even though the inner block variable is declared inside the outer block, the inner block's scope is restricted to the inner block.

1.8 Video

- BJP 2-4, *Nested for Loops*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c2-4
- CSE 142, *Nested for Loops* (16:18–37:50)
<https://www.youtube.com/watch?v=oeUm1RFGkWw&start=2524>

1.9 Forum discussion

Lesson 2.06 Nested for Loops (TEALS Discourse account required)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 2.08 – Programming Project

1.1 Overview

1.1.1 Objectives – *Students will be able to...*

- **Plan and construct** a structured program containing nested loops.

1.1.2 Assessments – *Students will...*

- **Submit** a complete, functional program by the end of next class

1.1.3 Homework – *Students will...*

- **Read** BJP 2.4 “Class Constants”
- **Outline** Chapter 2, omitting BJP 2.5

1.2 Materials & Prep

- **Projector and computer**
- **Student self-help system** (such as C2B4 or student pairing)

Make sure you are set up to grade student notebooks today. If possible, you should only collect 3 – 5 notebooks at a time so students have their notebooks available to reference during programming time.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction & classroom procedures	10min
Programming project	30min
Students trade work, check, & submit	10min

1.4 Procedure

To prepare students for the upcoming unit exam, the next few class periods will be devoted to reinforcing concepts and applying the tools, procedures, and code that were introduced this unit.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Classroom Procedures [10 minutes]

1. If your computer time requires you to move to another room or to change seating, you should teach and/or review those procedures before introducing the lab material. It's been a few weeks since the last long form programming assignments, so make sure to ask students what the procedures are if they:
 - have gotten stuck (check pseudocode and structure diagram),
 - finished early (move on to challenge questions), or
 - can't remember a coding rule or procedure (check your notes, worksheets, and textbook, C2B4)
2. Unless you have had students submitting work electronically regularly, you should model and review those procedures before students begin work.
3. Introduce the programming project, taking a moment to talk strategy with your class.

PROGRAMMING PROJECT: Write a program that produces the following figure as its output using nested for loops.

```
|*****|  
 \:::/  
  \::/  
   \:/  
    \/  
     /\  
      /::\  
       /:::\  
        /:::::\  
         /:::::::\  
          /:::::::|
```

//

TIPS: Start with a structure diagram or writing out steps in English as pseudocode. Try to isolate repeated tasks into methods. Include comments in with your code so others can easily understand what the code is supposed to do.

4. Ask your class for suggestions as to how to tackle this programming problem. Students should suggest drawing a structural diagram, building the program one method at a time (iterative development), and following the correction steps on their personal algorithms (debugging).

Procedural decomposition is hard! As a group, ask students to discuss what components go into drawing each line.

- What characteristics stay the same for each line? (Slashes, colons, spaces)
- What characteristics could we use a loop for?
- What might we want to make its own method that we can call more than once?

1.5

1.5.0.1 Emphasize with students...

1.5.0.2 Big Ideas - Applying the concept of a nested for-loop in drawing artistic patterns

A for-loops allow the repeated execution of one statement (or group of statements). As you iterate through the repetitions, a counter (variable) keeps track of how many times it has already done. The fun thing is: this counter value can also be used for a *nested* for-loop (ie, have one for-loop inside another for-loop). This combination of using two for-loops means you can write some code to repeat object/patterns that "grow" or "shrink".

You can think of it this way: A nested loop combination allows you to repeat execution of one statement ... but that one statement *itself* includes a repetition also.

As an alternative to the "hour-glass" picture, there are other alternatives that use nested for loops:

- a symmetrical Christmas tree, with a tree trunk (can add some decorations!)
- a series of 3 (possibly nested) pyramids (small, medium, large)
- a house with roof, a tall tower, or a rocket ship (symmetrical)
- a pattern on a fabric or carpet, or stained glass window
- nested shapes, or other optical illusion
- other ideas

In this ASCII art programming project, create something unique!

1.6

1.6.1 Programming Project [30 minutes]

Get students started on the ASCII art programming project. It can be the hour-glass picture, or something different! Offer students help after they have tried to answer the questions themselves:

- a. Have they checked the book for examples?
- b. Have they asked a friend (or two) for help?

If students seem to be getting stuck on the same segment of code, offer a hint or tip on the board (silently, without disrupting student flow).

If the entire class is stuck, return to whole group and work through the programming challenge together as a class, having students offer an increasing proportion of the answers as you move along.

1.6.2 Students trade work, check, evaluate, and turn in [5 minutes]

At the end of class, have students look over each other's projects before submitting.

Evaluation Question: How many lines of code did this program "save" by using for-loops? How can the code be modified for some other interesting effect, or embellishment?

1.6.2.1 Emphasize with students...

1.6.2.2 Curricular Competancies - Share and evaluate

It's fun to share artwork. It spurs the imagination. It's valuable to see how others apply the same concept.

Computers are great at doing repetitive tasks. Ask a few questions to evaluate each other's code. For example: How many lines of code did you save by using loops? A well-decomposed solution should result in clean, simple code, using as few lines as possible. How can the code be modified for some other interesting effect, or embellishment?

1.7 Accommodation and Differentiation

If you have students who are speeding through this project, you should encourage them to:

- Finish the programming project started in class yesterday.
- Act as student TAs and help struggling classmates (NOTE: you should specifically direct students NOT to give answers, but to help students think of ideas on their own.)

If you have students that are struggling during this class (and you will), resist the urge to help students too much at this stage. Ask leading questions, direct students to their notes, or an example that demonstrates a similar solution, but don't give students the answer here. Resilience/grit is an important emotional tool for solving complex programming problems: the emotional journey students take during these difficult programming problems is as important as the actual coding challenge.

If students are having trouble due to language, pair students up so those with more advanced English can help those that are emergent language learners.

1.8 Forum discussion

[Lesson 2.08 Programming Project \(TEALS Discourse account required\)](#)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 2.08a – Programming Project Alternative - Lyrics

1.1 Overview

1.1.1 Objectives – *Students will be able to...*

- **Plan and construct** a structured program containing nested loops.

1.1.2 Assessments – *Students will...*

- **Submit** a complete, functional program by the end of next class

1.1.3 Homework – *Students will...*

- **Outline** Chapter 2, omitting BJP 2.5

1.2 Materials & Prep

- **Projector and computer**
- **Student self-help system** (such as C2B4 or student pairing)

Make sure you are set up to grade student notebooks today. If possible, you should only collect 3 – 5 notebooks at a time so students have their notebooks available to reference during programming time.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction & classroom procedures	10min
Programming project	30min
Students trade work, check, & submit	10min

1.4 Procedure

To prepare students for the upcoming unit exam, the next few class periods will be devoted to reinforcing concepts and applying the tools, procedures, and code that were introduced this unit.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Classroom Procedures [10 minutes]

1.4.2.1 Classroom Procedures

1. If your computer time requires you to move to another room or to change seating, you should teach and/or review those procedures before introducing the lab material.
2. It's been a few weeks since the last long form programming assignments, so make sure to ask students what the procedures are if they:
 - have gotten stuck (check pseudocode and structure diagram),
 - finished early (move on to challenge questions), or
 - can't remember a coding rule or procedure (check your notes, worksheets, and textbook, C2B4)
3. Unless you have had students submitting work electronically regularly, you should model and review the procedures for submitting electronic work before students begin work.

Introduce the programming project, taking a moment to talk strategy with your class.

PROGRAMMING PROJECT: Write a program that prints out the lyrics of a simple, repetitive song using the minimum number of words in ``print/ln()`` statements by factoring repetitive words using nested for loops. Here are examples of two songs that can be used: "Row, Row, Row Your Boat", "Jingle Bells"

1.4.2.2 Song 1: "Row, Row, Row Your Boat"

Row, row, row your boat
Gently down the stream
Merrily, merrily, merrily, merrily
Life is but a dream

Row, row, row your boat
Gently down the stream
Merrily, merrily, merrily, merrily
Life is but a dream

Song 2: "Jingle Bells"

Jingle bells, jingle bells
Jingle all the way
Oh, what fun it is to ride
In a one horse open sleigh

Jingle bells, jingle bells
Jingle all the way
Oh, what fun it is to ride
In a one horse open sleigh



1. In computer science, we look for these patterns as ways to factor out repetition in order to gain efficiency. Students may have other childhood songs from their background that reflects their culture. You can have a discussion on what songs they grew up with and look for patterns in the song. The two songs are just examples. At the teacher's discretion, the students may substitute a song from their childhood. Other songs that may be used are "Happy Birthday" or "You Are My Sunshine".

TIPS: Start with a structure diagram or writing out steps in English as pseudocode. Try to isolate repeated tasks into methods. Include comments in with your code so others can easily understand what the code is supposed to do.

2. Ask your class for suggestions as to how to tackle this programming problem. Students should suggest drawing a structural diagram, building the program one method at a time (iterative development), and following the correction steps on their personal algorithms (debugging).

Procedural decomposition is hard! As a group, ask students to discuss what components go into drawing each line.

- What words repeat?
- What words or phrases repeat on multiple lines?
- What might we want to make its own method that we can call more than once?

1.4.2.3 Emphasize with students

1.4.2.3.1 Big Ideas - Applying the concept of a nested for-loop in drawing artistic patterns

A for-loops allow the repeated execution of one statement (or group of statements). As you iterate through the repetitions, a counter (variable) keeps track of how many times it has already done. The fun thing is: this counter value can also be used for a *nested* for-loop (i.e., have one for-loop inside another for-loop). This combination of using two for-loops means you can write some code to repeat object/patterns that “grow” or “shrink”.

You can think of it this way: A nested loop combination allows you to repeat execution of one statement ... but that one statement *itself* includes a repetition also.

As an alternative to the “Row, Row, Row Your Boat”, “Jingle Bells” song, there are other alternatives that use nested for loops:

- Happy Birthday
- You Are My Sunshine
- There was an Old Lady Who Swallowed a Fly
- Feliz Navidad
- Songs from student's childhood

1.4.3 Programming Project [30 minutes]

The programming project portion has a number of different options.

1.4.3.1 Option 1 Print it as a round

“Row, Row, Row Your Board” is frequently sung in a round. This challenge would be to use the factored song, in this case “Row, Row, Row Your Boat” and print out two sets of lyrics for singing in a round. The following is one round but the project could be extended to generalize to N rounds. Note students do not know how to get input yet so the N would be fixed.

¹ Row, row, row your boat
²

¹ Gently down the stream
² Row, row, row your boat

¹ Merrily, merrily, merrily, merrily
² Gently down the stream

¹ Life is but a dream
² Merrily, merrily, merrily



1.4.3.2 Option 2 - Alternative Lyrics

1.4.3.2.1 Part 1

Have students can look up the entire lyrics to their song and factor the entire song.

1.4.3.2.2 Part 2

1.4.3.3 Option 3:

Students can make their own song using the same music. Have students think up thier own alternative lyrics and substitute it in the print statements to create their own song. For example, substituting “drive” for “row” and “car” for “boat”, the program would now generate “drive, drive, drive your car”. Students can pick a popular song they know the chorus to, factor the song in Java, and substutute their own lyrics.

1.4.4 Bonus:

As a bonus to the project, print out the number of words for each line:

Row, row, row your boat (5)
Gently down the stream (4)
Merrily, merrily, merrily, merrily (4)
Life is but a dream (5)



1.4.5 Students trade work, check, evaluate, and turn in [5 minutes]

At the end of class, have students look over each other’s projects before submitting.

Evaluation Question: How many lines of code did this program “save” by using for-loops? How can the code be modified for some other interesting effect, or embellishment?

1.4.5.1 Curricular Competencies - Share and evaluate

It's fun to share artwork. It spurs the imagination. It's valuable to see how others apply the same concept.

Computers are great at doing repetitive tasks. Ask a few questions to evaluate each other's code. For example: How may lines of code did you save by using loops? A well decomposed solution should result in clean, simple code, using as few lines as possible. How can the code be modified for some other interesting effect, or embellishment?

1.5 Accommodation and Differentiation

If you have students who are speeding through this project, you should encourage them to:

- Finish the programming project started in class yesterday.

- Act as student TAs and help struggling classmates (NOTE: you should specifically direct students NOT to give answers, but to help students think of ideas on their own.)

If you have students that are struggling during this class (and you will), resist the urge to help students too much at this stage. Ask leading questions, direct students to their notes, or an example that demonstrates a similar solution, but don't give students the answer here. Resilience/grit is an important emotional tool for solving complex programming problems: the emotional journey students take during these difficult programming problems is as important as the actual coding challenge.

If students are having trouble due to language, pair students up so those with more advanced English can help those that are emergent language learners.

1.6 Forum discussion

[Lesson 2.08 Programming Project \(TEALS Discourse account required\)](#)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 2.09 — Programming Project

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Ideate, plan and construct** a structured program containing nested loops.
- **Share and evaluate** one another's code

1.1.2 Assessments — *Students will...*

- **Submit** a complete, functional program by the end of class.

1.1.3 Homework — *Students will...*

- **Complete** practice questions with class constants

1.2 Materials & Prep

- **Projector and computer**
- **Student self-help system** (such as C2B4 or student pairing)

Today is day 2 of the programming project; be sure to remind students that they only have the first 30 minutes of class to finish and submit their program. Be prepared to review the correct code and offer an upgrade using class constants for the second half of class.

If you have not finished grading notebooks, make sure you are set up to grade student notebooks today. If possible, you should only collect 3-5 notebooks at a time so students have their notebooks available to reference during programming time.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Programming project	30min
Introduction to class constants	10min
Students update their code	10min

1.4 Procedure

Today's lesson will be a combination of drilling the parts of a basic program, and conditioning students to check for common errors. To hook your class, have pictures of punch cards and punch card readers up when students enter. If possible, have physical punch cards available to pass around the room for tactile learners as you explain the origins of the phrase "bug" and "debugging."

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Programming Project [30 minutes]

Have students continue the programming project, aiming to finish with about ten minutes left in class.

1.4.3 Introduction to Class Constants [10 minutes]

1. Present on a board or projector a complete Java class that solves the programming project. Ideally, this would be a student solution. Walk the class through the solution, having students help you trace the flow of control and predicting output to confirm that the program works.
 - o Suggest that there might be an easier way to update repetitive values at once, and introduce **class constants**.
2. Trace or run the program with the updated class constants to demonstrate that this does indeed work.
3. If you are running the program in IDE, show that you can easily change the constants. Change the values several times, running the program each time to drive this home.

1.4.4 Students Update Their Code [10 minutes]

Give students time to update their project code, now including class constants.

1.5 Accommodation and Differentiation

If you have students who are speeding through this project, you should encourage them to act as student TAs and help struggling classmates (NOTE: you should specifically direct students NOT to give answers, but to help students think of ideas on their own.)

If you have students that are struggling during this class (and you will), resist the urge to help students too much at this stage. Ask leading questions, direct students to their notes, or an example that demonstrates a

similar solution, but don't give students the answer here. Resilience/grit is an important emotional tool for solving complex programming problems: the emotional journey students take during these difficult programming problems is as important as the actual coding challenge.

If students are having trouble due to language, pair students up so those with more advanced English can help those that are emergent language learners.

1.6 Forum discussion

Lesson 2.09 Programming Project (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 2.10 — Finding & Fixing Errors

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Find** errors in their returned homework assignments.
- **Correct** their code

1.1.2 Assessments — *Students will...*

- **Re-submit** all homework assignments with corrected answers.

1.1.3 Homework — *Students will...*

- **Study** for the test by:
 - **Reviewing** all the blue pages at the end of Chapter 2
 - **Re-reading** sections as needed
- **Submit** 5 questions for review in class tomorrow using electronic survey

1.2 Materials & Prep

- **Any student homework assignments** that you have not yet returned
- **Student self-help system** (such as C2B4 or student pairing)
- **Electronic survey** for student review requests

When you grade homework assignments, it will be most useful to these lessons if you only mark an answer incorrect or correct. ELL classrooms are the exception to this rule—students will be having a hard enough time just reading the material; you can speed along their processing by correcting one example, then having them look for similar errors with that example.

The homework tonight asks students to submit 5 questions for review. Create an **electronic survey** for students to complete with 6 text fields, one for name, and 5 for questions they have about Ch. 2 content. Set a time-deadline (e.g. 10pm) by which time students must have submitted 5 questions from Ch.2 that they would like to see reviewed in tomorrow's class. If students do not have questions, stipulate that they still have to submit something to receive credit, even if it is only questions they think other students may have.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction and homework distribution	5min
Student work	35min
Students trade work, check, and submit	10min

1.4 Procedure

Today we continue reinforcing concepts and applying the tools, procedures, and code that were introduced last week. Students will have the opportunity to correct any incorrect homework assignments. If students did not have time to finish the programming projects from yesterday, you may allow them time to work on those projects today.

This is a good day to loosen up the vibe in the classroom a bit. Try playing music softly in the background to encourage students to relax and focus on spotting errors. Try to avoid loud, rhythmic music, and avoid the pitfall of allowing students to select the station!

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Homework Distribution [5 minutes]

1. Return student homework packets, or have students place their returned homeworks in a pile on their desk.
2. Explain to students that they have the opportunity to get full credit on their homework grades by correcting them now, in class. Ask students for suggestions/ideas on how to make sure they don't miss any errors.
 - a. By now students should be used to relying on their error checklist/algorithm.
 - b. Hopefully, students suggest using the 4 Commandments of Scope, creating pseudocode and/or structure diagrams to clarify thinking on program structure, and checking their notes and the text book.

1.4.3 Student Work [35 minutes]

Have students work individually to correct their homework grades.

- Offer time checks for students so they stay on task.
- If students have not finished their programming project from yesterday's class, allow them to do so today.

1.4.4 Students trade work, check, and turn in [10 minutes]

At the end of class, have students trade their homework assignments to evaluate each other's corrections before submission.

1.5 Accommodation and Differentiation

If you have students who are speeding through this lesson, you should encourage them to tackle programming project 2 & 3 in the text book.

If you were unable to finish grading student notebooks yesterday, finish them today while students are working. Return notebooks by the end of class so students may use them to study for the exam.

1.6 Forum discussion

[Lesson 2.10 Finding & Fixing Errors \(TEALS Discourse account required\)](#)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 2.11 – Review

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Identify** weaknesses in their Unit 2 knowledge.

1.1.2 Assessments — *Students will...*

- **Create** a personalized list of review topics to guide tonight's study session.

1.1.3 Homework — *Students will...*

- **Study** for tomorrow's test! Review ch 2 (omit 2.5)

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and marker**
- **Results** from electronic survey of review topics
- **Classroom copies** of the practice test [WS 2.11](#)

Once students have submitted their review requests, assemble those topics into categories and prepare to re-teach the topics as needed.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction and test format orientation	15min
Test review	30min
Check student study lists	5min

1.4 Procedure

Engage the class in the review session by pointing out that your review topics have been hand selected by the class. Explain that you will review test-taking strategies in addition to reviewing subject matter.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Test Format Orientation [15 minutes]

1. Clearly indicate that you expect all students to have a list of review topics to study this evening. Periodically remind students that this list will be checked at the end of class.
2. Students should already be familiar with the sections of the test, but it doesn't hurt to have students re-read the directions.
3. Work through the sample problems on the test as a way of reviewing topics, and answer any questions that students bring up as you go.

1.4.3 Test Review [30 minutes]

1. Using the results from the electronic survey, address the various review topics, prioritizing questions that popped up the most.
 - a. Some questions you may already have addressed while working through the sample test.
 - b. Be ready for additional questions to pop up as you go. Save yourself the work and use old homework questions and student-generated test questions as examples to work through.
 - c. Jot down notes about which topics you covered in review so you can adjust the exam to reflect the topics your students have learned.
2. Use a combination of group-solving questions on the whiteboard, think-pair-share, and timed-response as review strategies.
3. After you've completed reviewing an idea, remind the class that they should write down that topic if they feel they still have to review it tonight. (Yes, this will be a reminder every few minutes, but it will pay off later when students start creating review lists without prompting later in the year!)

1.4.4 Check student study lists [5 minutes]

Spend the last 5 minutes of class checking each student's review topic list.

1.5 Accommodation and Differentiation

In ELL classes, you may want to change code-writing questions to Parsons Problems. Educational research shows a high correlation between Parsons scores and code writing scores, and a low correlation between code writing and tracing and between Parsons and tracing. (In other words, Parsons Problems accurately assess students' ability to create code.) For more information on Parsons Problems, check out this paper (<https://cseweb.ucsd.edu/classes/fao8/cse599/denny.pdf>).

Even in a non-ELL class, you may want to change some Section II questions to Parsons problems because (1) grading the questions is easier, since logic and syntax errors are easy to discern, and (2) students challenged by language processing are able to more quickly complete the problem.

If your students are easily completing the programming projects in the week leading to the test, you may want to edit the test by deleting the "fill in the blanks" and leaving empty space.

1.6 Forum discussion

[Lesson 2.11 Unit 2 Review \(TEALS Discourse account required\)](#)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.00 — Test Review & Reteach

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Re-learn or strengthen** content knowledge and skills from Unit 2.

1.1.2 Assessments — *Students will...*

- **Re-submit** test answers with updated corrections for partial or full credit
 - Credit depends on instructor preference

1.1.3 Homework — *Students will...*

- **Correct** any incorrect test answers by re-answering on a separate sheet of paper
 - To get back credit, they must justify their new answers
 - Staple new answer sheet to old test and turn in tomorrow

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Corrected student tests**
- **Student grades** (posted online, emailed to students, or handed back on paper in class)
- **Digital copy of test questions** for projector

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Class discussion (if needed)	10min
Test review and reteach	35min
Check student notes and return tests	5min

1.4 Procedure

Return student grades before class begins or while students are completing the bellwork.

Do not return students' tests before the review session, since you want to motivate students to pay attention to the entire review, taking supplemental notes the entire time.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Class Discussion (if needed) [10 minutes]

1. If grades are low, invite the class to a discussion of what can be improved. Begin with student complaints and suggestions to build student buy-in. Ask students:
 - how they felt they were going to do before the test
 - what surprised them once they were taking the test
 - what they felt worked in the first unit (lessons, review strategies, assignments)
 - what do they think they want to change for the second unit
2. Once you feel that a dialogue has been established, validate students' feelings, then challenge them (e.g. AP courses are stressful, but this is good practice for college, where the pace is faster and professors don't give personalized instruction). In a non-judgmental, supportive tone, remind students that to be successful in the course:
 - Reading is mandatory
 - Homework is mandatory (And valuable! You will never assign "busy" work.)
 - To better manage their time, students should plan for 1 hour of homework a weeknight, with up to 2 hours of homework each weekend. If this seems impossible, they should meet with you or their guidance counselor to assess whether they can fit in an AP class at this time.
 - It is VERY important to keep your tone sympathetic at this point—an overworked, overstressed, underperforming student will slow your entire class down, and color that student against CS for the future!

1.4.3 Test Review and Reteach [30 minutes]

1. Walk the students through each question on the test, glossing over questions that everyone answered correctly.
 - a. You can ask for students to volunteer answers, or call on students randomly. Make sure that students explain their logic when they answer. If a student gives an incorrect answer, the explanation will tell you what you need to re-teach or clarify.

- b. Do not skip questions that everyone answered correctly, but do not spend more than the time it takes to read the question, and congratulate students' correct answers.
2. Project a copy of each question as you review—this will help students recall the question/process the information.
3. Make sure that students are taking notes during the re-teach, reminding students that for homework, they will have an opportunity to win back some of the points on their exam.
4. For Section II questions, select a sample of student work (with any identifying information obscured), and work through the answer together as a class.

1.4.4 Check student notes and return tests [5 minutes]

At the end of class, check student notes, and return the tests in hard copy form if applicable.

1.5 Accommodation and Differentiation

If students' grades are suffering because the reading assignments are taking them too long, you have a few options (some more drastic than others):

- Set aside classroom time to read through the assignment before students leave.
- Give students the lines of code needed to complete assignments, but in jumbled order. Have students rearrange the lines of code into the proper program (this is called a Parsons Problem).
- Flip your classroom: record your lectures, and have students watch them and take notes for homework. Any classwork drills or worksheets can be distributed for "homework," and the more complicated assignments that would normally be done at home, can be completed with your help when they come to class.

Create printed-out sheets instead that students can write code onto. Class time should then be filled with reading assignments, and more complicated coding practice so that you are available to tutor as needed.

- Encourage advanced students to take on additional programming challenges. One easy way to do this is to assign Programming Projects from the blue pages at the end of each Chapter. A more open-ended (and more interesting) challenge would be to have students flesh-out additional sections of a larger, year-round Pokémon-esque game. For each concept you learn, ask your advanced students to think of a feature or sequence from the game that can be programmed using the tools they've acquired.
- Encourage students to carefully name their files, and leave lots of comments so they can use the code later in the year to put the game together.

1.6 Forum discussion

[Lesson 3.00 Test Review & Reteach \(TEALS Discourse account required\)](#)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.01 – Parameters

1.1 Overview

1.1.1 Objectives – *Students will be able to...*

- **Correctly construct** formal and actual parameters (arguments).
- **Predict** output of programs that use parameters

1.1.2 Assessments – *Students will...*

- **Teach** a mini-lesson explaining the relationship between parameters and values
- **Submit** Practice questions

1.1.3 Homework – *Students will...*

- **Read** BJP 3.1 “Limitations of Parameters”
- **Complete** self-check questions 4-7

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Assignments** for 5 student groups
- [Animated GIF of software development gone awry]()

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Instruction	15min
Student mini-lesson preparation	10min
Student mini-lesson delivery	10min
Practice Activity	10min

1.4 Procedure

Your hook for today is an example of what happens when we don't stay flexible with our programming. Have the gif up and looping, and explain that this image is an illustration of what can happen to our programs if we don't make them adaptable.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Instruction [15 minutes]

1. Begin class with a lecture/discussion that introduces parameters
 - o Part of the reasons that humans are so good at solving new problems is that they generalize the solution to whole categories of problems.
 - o The book gives an example of walking as a generalizable task—walking 20 steps and walking 10 steps can be described as the task of “walking forward,” and the part that varies (we call this a parameter) is the number of steps.
 - In programming, when we make code flexible by “**parameterizing**” parts of the task that are likely to change, we end up with programs that are shorter, easier to understand for other coders, better organized, and reusable.
 - Imagine how difficult life would be if you had to separately learn all the movements required to walk 10 steps down the runway in addition to the procedure for walking 20 steps it takes to get to the stage? Instead, your brain computes a general rule something like “walk only the number of steps to the next obstacle,” then your eyes and ears input how many steps that should be.
 - Java is the same way—it takes less memory and computing power to execute a program if you write code that is flexible/reusable with different parameters. (In fact, in this chapter we’re going to learn how to use user input—like the information we get from our eyes and ears—to influence the behavior of the programs we write!)
 - What are some other behaviors that we “parameterize” every day? (If students need help getting started, suggest:
 - Braiding hair: different heads require a different number of braids, but the braiding is always the same procedure.
 - Turning on the stereo or TV: the methods to turn on the TV, radio, or stereo are always the same, but you adjust the volume to different levels depending on the time of day. What would the parameters be in this example? (Time, volume)

2. Review this programming example from the textbook:

- In the last chapter, we inserted spaces into a drawing by calling a method:

```
writeSpaces();
```

//

- If we wanted to tell method writeSpaces to output 10 spaces, we might decide to declare a variable:

```
int number = 10; // Can anyone explain why this won't work?  
writeSpaces();
```

//

→ This won't work because number's scope is outside the writeSpaces method.

- Instead we parameterize the number of spaces by changing the method header as highlighted

```
public static void writeSpaces (_int number_) {  
    for (int i = 1; i <= number; i++) {  
        System.out.print(" ");  
    }  
}
```

//

- Now when we call the method, we can (and must!) include the parameterized value:

```
writeSpaces(10);  
writeSpaces(); // ERROR: No parameter specified  
writeSpaces(24901);  
writeSpaces(9/3-2);
```

//

- On the board or in IDE, point out the difference between an actual parameter (argument) and a formal parameter (or just "parameter"). Point to a few different examples of each on sample code, asking students which one is which, and how they know.

1.4.3 Student Mini-Lesson Preparation [10 minutes]

1. Students are going to get a chance to teach the second half of class—the mechanics of parameters. Assign each group one section of the sample code in section 3.1 "Mechanics of Parameters."

Student groups should take 10 minutes to review the program, read the example on the pages following the example, then figure out how they want to explain to class what values are being stored in the computer's memory.

2. On the board or overhead, give students a few things they should consider in planning their mini lesson:

- a. Who is going to speak when?
- b. How are you going to illustrate the flow of control?
- c. What do you need to have up on the board to illustrate your mini-lesson, and who oversees writing it out?
- d. Where and how will you feature the output produced by your code segment?

1.4.4 Student Mini-Lesson Delivery [10 minutes]

Have student groups sequentially teach through the example in the book, demonstrating the changes to the stored value, predicting output, and tracing the flow of control.

1.4.5 Practice Activity or Homework [10 minutes]

Have students to work individually on practice self-check problems.

- a. Self-Check 3.1: methodHeaderSyntax b. Self-Check 3.2: MysteryNums c. Self-Check 3.3: Oops3-errors

1.5 Accommodation and Differentiation

If students are struggling with content, you might opt to work on the practice problems as a whole group. In some classes, the teaching exercise might take an entire class period.

- To keep from losing too much instructional time, remind students of their presentation time limit while they are planning, and during their presentation.
- Check students as they are coordinating their mini-lesson; offer feedback on the timing of their lesson.
- Use a timer during student lessons, and hold up a “1 minute” warning sign to keep students on pace. This can become a fun challenge. Use a buzzer or gong to keep the lessons on schedule in a non stressful way.

If students are speeding through the content, encourage them to complete one of the following projects (depending on how much time is available):

- Create a poster for the classroom that diagrams/illustrates how to correctly parameterize a method.
- Tackle Ch.3 Programming Project \#1 for extra credit. (This might be completed over the course of several days.)

1.6 Misconceptions

- When passing parameters to methods, students may include the type when the parameter is a variable:
`whiteSpace (int x);`

1.7 Video

- BJP 3–1, *Mechanics of Parameters*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c3-1
- CSE 142, *Single Parameters* (21:05-33:05)
<https://www.youtube.com/watch?v=otR7yEbRZAs&start=1265>
- CSE 142, *Multiple Parameters* (33:06-50:0)
<https://www.youtube.com/watch?v=otR7yEbRZAs&start=1985>
- CS Homework Bytes, *Functions and Parameters, with Komal*
<https://www.youtube.com/watch?v=UuZCQWErV-A>

1.8 Forum discussion

Lesson 3.01 Parameters (TEALS Discourse account required)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.02 — Limitations of Parameters & Multiple Parameters

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Modify** programs using parameters and class constants to create original artworks.

1.1.2 Assessments — *Students will...*

- **Complete** an art project and “artist statement” justifying their programming choices

1.1.3 Homework — *Students will...*

- **Read** BJP 3.1 “Limitations of Parameters”, “Multiple Parameters”, “Parameters versus Constants”
- **Jazz up** art projects and programs

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Sample** of final project (picture, artist statement, code)
- **Pictures** from previous year’s “gallery opening”
- **Ball** for passing flow of control

Around this time of year, many schools have open houses and/or parent-teacher conferences. If you have the time and resources, this programming project is a excellent opportunity for students to showcase their work for parents, principals, superintendents, etc. For instructions on using this lesson as an opportunity to share student work, refer to Accommodation & Differentiation at the end of this lesson plan.

If this is your first year teaching this lesson, you obviously won’t have pictures from previous years events. Make sure to document this lesson for future years so you can hook students with the glamour and excitement of an art gallery opening.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Review and intro to multiple parameters	15min
Student practice	30min

1.4 Procedure

Hook your students by showing an example of a completed art project, statement, and code sample. If you don't have previous work to share with students, show them some samples of ASCII art and ask students to predict how many lines of code they'd need to design the images.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Review and Introduction to Multiple Parameters [15 minutes]

1. If you will be holding a reception for student art (see end of lesson for details), announce that now. Segue into today's lesson by explaining that you're going to give students a few additional tools to use in their art project. Give them a quick recap/review before teaching **multiple parameters**:

- o Call students up to the board—some to help you demonstrate flow of control, another to produce console output on the whiteboard.
- o Using the example below, review the flow of control by having the students (main and parameter) pass information to you (the method). Have the class direct your other student what output to write on the board. Ask students what variable x is throughout the example to drive home the idea that parameters don't change value when the method changes a local value.

```
public class ParameterExample {
    public static void main (String[] args) {
        int x = 17;
        doubleNumber(x);
        System.out.println("x = " + x);
        System.out.println();
        int number = 42;
        doubleNumber(number);
        System.out.println("number = " + number);
    }
    public static void doubleNumber (int number) {
        System.out.println("Initial value = " + number);
        number = number * 2;
        System.out.println("Final value = " + number);
    }
}
```



- o It is possible to declare multiple parameters! The trick is to always make sure your method accepts the parameters in the same order. When calling the method, pass the parameters in the same order in which they were declared.

```
public static <type> <name> (<type> <name>, <type> <name>, ... <type> <name>) {
    <statement>;
    <statement>;
    ...
    <statement>;
}
```



2. Using the diagram below, walk students through this coding example. You should have students draw, circle, and explain the parts of the code in their own notebooks so they have a chance to think this through.

```
public static void main (String [] args) {  
  
    writeChars ('=', 20); //----- Writes a line of 20 '='s  
    System.out.println(); //----- Returns to the next line  
  
    for (int i = 1; i <= 10; i++) { //----- For 10 lines of picture (height)  
        writeChars('>', i); //----- Increase the number of '>'s in each line  
        writeChars(' ', 20 - 2*i); //----- Decrease the number of spaces in each line  
        writeChars('<', i); //----- Increase the number of '<'s in each line  
  
        System.out.println(); //----- Go to the next line before starting the body  
        // of the loop again.  
    }  
  
    public static void writeChars (char ch, int number) {  
        for (int i = 1; i <= number; i++) {  
            System.out.print(ch);  
        }  
    }  
}
```

- Have students predict how the output will change if you change 1 or 2 things in the code, then allow them to start their open-ended activity.



1.4.3 Student Practice [30 minutes]

1. Encourage students to spend a few minutes fiddling with the code we reviewed in class to see how it changes the outputted image. It's really important that students have time and space to fiddle with code to understand how the different parts relate. Encourage students to start with this code and build from it/change it to make whatever images they want to make for their final project.
2. Instruct students that they will need to create a design that uses parameters, loops, and/or nested loops. Extra points will be awarded for particularly complex/creative images.
3. They should use (and keep copies of) a structure diagram and/or pseudocode for their own records, since they will be expected to write a short paragraph explaining in plain English (or their native tongue).
4. They are allowed to continue working on this project outside of class: you have graciously decided not to assign problems for homework tonight so that they will have time to work on the code.

If you find that students are highly engaged in the project, you might opt to extend this lesson to a 2 period coding session. The more opportunity students have to manipulate code and check output, the more intuition they will have down the road as they internalize what individual pieces of code mean.

1.5 College Board Topic Questions

After this lesson, students will be able to answer questions from the College Board Unit 2 Topic Questions 2.9 Using Math Class

1.6 Accommodation and Differentiation

If you feel that students need the extra review, ask students to help you build the example program you use during the introduction of class. Do this as a whole group to keep class on pace, and write only the information that students give you (i.e. let mistakes happen, then guide the class to error-check their own work).

If students are having a hard time with the coding project, you may encourage them to work in pairs instead of as individuals. If some of your class partners up, make sure that the class understands that teams will be expected to write more complex images than those created by students working alone.

Encourage students to think about their learning and coding processes by having students articulate HOW they've learned what they learned. This can be done by having students explain their coding decisions to peers and laypeople with no coding experience. A great way to do this in a low-pressure setting is to have students share their work with parents and others at an open house; ham it up a little for fun!

- Display student work around the room, and have students explain or talk about their work during an "art gallery opening" held at lunch, after school, or in the evening during parent teacher conferences or open house events.
- Invite your principal, superintendent, and school staff to attend—this is a great way to increase community buy-in for your class.
- Provide (or have students bring in) cubed cheese, fruit, crackers, white grape juice, and crackers. Create a fancy reception table with a table cloth, plastic wine glasses, and cocktail napkins.
- Play jazz, string quartet, or trip hop softly in the background to create a trendy reception vibe, and have students "dress like artists" and stand near their work. Make sure you dress up/down to fit the scene!
- If possible, hand visitors "question cards" to give them ideas of how they can learn more about student work as they circle the "gallery." Cards should include prompts like:
 - Tell me about your picture; how did you come up with this idea?
 - Can you explain to me a bit about how you get the computer to draw this image?
 - How did you go about the process of writing this code?
 - How did you start this project?
 - If I wanted to change this picture, how would I go about doing it?

An event like this maintains/raises student morale as students:

- take ownership of their work,
- receive praise for their hard-earned accomplishments thus far, and
- realize how much they've learned when they explain code to a non-coder.

1.7 Forum discussion

Lesson 3.02 Limitations of Parameters & Multiple Parameters (TEALS Discourse account required)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.04 — Programming Project

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Write** a program that uses parameters, the math class, and returns values.

1.1.2 Assessments — *Students will...*

- **Submit** an Equestria program by the end of class.

1.1.3 Homework — *Students will...*

- **Complete** Ch.3 self-check questions 18 & 19

1.2 Materials & Prep

- **Projector and computer**
- **White paper and markers**
- **Classroom copies** of [WS 3.4, Equestria](#)
- **Classroom copies** of Algorithm for Solving Problems
- **Poster or image** of Equestria map with Cartesian coordinates (Poster 3.4)

The handout “Algorithm for Solving Problems” should be used/drilled every time the students are asked to solve a larger, AP Test Section II – type problem. You may find it useful to make a large-format poster to hang in your room so students always see the steps they should use to tackle a large programming problem.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction to programming project	5min
Student programming time	45min

1.4 Procedure

Over the next few weeks, we'll be introducing students to larger programming projects to meet the AP Computer Science A lab requirements. It is in your and your students' best interest if you encourage them to help themselves and each other before seeing you. Depending on the length of your class periods, this lab may take 2 class periods to complete.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction to Programming Project [10 minutes]

Begin with an introduction to today's programming project.

- Students should complete this programming project individually. As a whole group, review the Algorithm for Solving Problems sheet. Read the sheet out loud, and briefly model the steps as you review the programming assignment. Read the programming assignment to the class, taking time to pause between each of the three requirements outlined in the question.
- Ask students what their very first steps should be according to the problem-solving algorithm.
- Require that pseudocode/comment documentation be submitted. Include these as part of the lab grade.
- Remind students to tackle one part of the problem at a time. It is OK if they leave pseudocode in while they solve a different part of the problem, and they don't have to solve all of parts of the program in any particular order.
- They should refer to their notes, textbooks, and posters/work around the room to help them come up with ideas/solutions to programming problems.
- Sometimes it's a good idea to step away from a problem for a few minutes if they're stuck—work on a different method and return to where they were stuck later.

1.5 Student Programming Time [45 minutes]

In an email, on the projector, or as a handout (WS 3.4), give students the following questions to work on individually, or in pairs.

1.5.0.1 Emphasize with students...

1.5.0.2 Content - Problem decomposition - Management of complexity

By the end of this activity you will have created quite a complex program. But notice how the activity takes you through the smaller steps in building the program?

The finished program is broken up into smaller steps, then they're each completed individually. When they're put together they create the larger program. This type of decomposition allows the programmer to manage complexity. Rather than considering the whole problem and program, the programmer can focus on each small component, then assemble them together at the end.

Writing programs with methods allows us to divide the program up into smaller parts quite easily.

1.5.0.3 PROGRAMMING PROJECT

1.5.0.3.1 Exercise 1

Princess Luna and Celestia are going on a tour of the kingdom to greet the other citizens of Equestria. Their tour takes them on a circular path (shown on map). Write a method called `roadTrip` that (1) accepts as a parameter the diameter of the circular path, and (2) returns the length of the trip.

The equation for circumference is: $C = \pi d$

Java has a math constant called `Math.PI`.

1.5.0.3.2 Exercise 2

Write a method called `distance` that (1) accepts four integer coordinates: x_1, y_1, x_2, y_2 as parameters, (2) computes the distance between points (x_1, y_1) and (x_2, y_2) on the map, and (3) returns that distance.

The equation for the distance is: $\sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$.

Test out your program by writing a main method that calls the `distance` method for each of the following pairs of cities. Your main method should output the value returned by the `distance` method.

1. Distance from Baltimare to Manehattan =
2. Distance from Los Pegasus to Neighagra Falls =
3. Distance from the Badlands to Ponyville =

1.5.0.3.3 Exercise 3

Write a program that helps Princess Luna plan a 3-stop tour of Equestria. Choose any 3 locations in Equestria, as defined by their x and y coordinates on the map. Your program should output the distance between the three destinations.

You should use the `distance` methods you wrote for Exercise 2.

1.5.0.3.4 Exercise 4

Write a method called `totalTrip` that accepts parameters for 3 locations (each containing coordinates) and returns the total distance traveled by visiting all 3 locations and returning to the starting location. You should use the `distance` methods you wrote in Exercise 2 and you can choose any 3 locations in Equestria.

Extra credit: make this program compute 4 locations instead!

1.6 Accommodation and Differentiation

For students who complete the lab early, ask them to flesh out their program by adding a method that will calculate the sum of the distances travelled between three cities in Exercise 2.

If you suspect that students will struggle with procedural decomposition, have them work in groups to figure out a course of action before beginning. Procedural decomposition is one of the hardest skills for students to acquire, so it is critical that you allow students to fight through the process.

If additional scaffolding is needed, you might list all of the parts of the program, or have students come up with the parts ("figure out how to write the equation in Java using the math class," "create some parameters to pass to a return method," "write the framework for a method to return the distance value") out of order, then give the students some time to organize the steps themselves. As a whole group, you can then come to consensus on what steps need to be approached, and what order components should appear in the final program.

1.7 Forum discussion

Lesson 3.04 Programming Project (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.06 — Interactive Programs & Scanner Objects

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Write** programs that accept user input using a scanner object.

1.1.2 Assessments — *Students will...*

- **Complete** Practice problems

1.1.3 Homework — *Students will...*

- **Read** BJP 3.3 “Interactive programming” and “Sample interactive program”
- **Outline** Chapter 3, except for BJP 3.4

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction to interactive programming	15min
Student practice	35min

1.4 Procedure

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction to Interactive Programming [10 minutes]

1. Hook the students with a brief discussion about Pokemon. Be sure that the following points come up during discussion (either you bring them up, or you guide the students to bring these points up themselves):
 - EV (experience values) are gained through combat. When one Pokemon wins against another, they win EV points to raise your stats.
 - These stats are private, the trainer (user) cannot see them.
 - If you want to know what your EV value is (so you can determine strategy and gain the most points), you need to compute it yourself given the values that you can see.
 - The formula for calculating HP is on Bulbapedia, and if we do some algebra, we can calculate EV. Let's write a program that will help us calculate EV for any Pokemon stats a trainer enters.

2. Have a student (or sequential students) come to the board to write the console output as you work through the example below. You should have students write and label this sample program as you write it. While discussing the different parts of the program, circle the room, checking to make sure students are keeping up with the notes.

- Write a method that reads user input for known Pokemon stats to determine EV (effort value).

```
import java.util.*; //
```

- Anytime you're getting user input, start with this import declaration:

```
public class GetEV {  
    public static void main(String[] args) {  
        Scanner userInput = new Scanner (System.in); //
```

- So far, the headers look familiar. When constructing the scanner:
 - You always capitalize Scanner
 - Name the scanner something useful (scanner, console, userInput)
 - Construct with the new keyword, and System.in.

- System.out.println ("This program calculates user EV.");
System.out.println ("Input your Pokemon's stats below:"); //

- This outputs to the console a message explaining the program to the user:

```
System.out.print ("Hit points: ");  
int hp = userInput.nextInt(); //
```

- This pairs a user prompt with a variable hp that holds the information the user inputted.

```
System.out.print("Level: ");  
int level = userInput.nextInt(); //
```

- Again, we pair the prompt for the user input with the scanner that accepts and stores that data as a variable for later use.

```
System.out.print("IV: "); // This is for initial value of the hit point stat.  
int iv = userInput.nextInt();  
System.out.print("Base HP: ");  
int base = userInput.nextInt(); //
```

- Next we need to write the formula that will calculate EV from the user input, then return a value (otherwise the scanner has no function!) You should have the students build as much of this formula as possible:

$$EV = (((HP - 10) * 100) / Level - 2*Base - IV - 100) * 4$$

```
        int ev = (((hp-10) * 100)/level - 2 * base - iv - 100) * 4;
        System.out.println("You have " + ev + " effort value points for your HP stat.");
    }
}
```

- If students need another example, work through the book example for mortgage payments, having students write more of the code than in the previous example.

1.5

1.5.0.1 Final Project

Pay close attention to the formulas used when calculating experience points and other values in the Pokemon game. The final project in this course will involve you creating a similar game to Pokemon, so these formulas will be very worthwhile to understand.

1.5.1 Student Practice [35 minutes]

1. Have students complete the following self-check questions:
 1. Self-Check 3.23: promptMultiplyBy2
 2. Self-Check 3.24: SumNumbers
 3. Self-Check 3.25: RobertPaulson
2. Have students complete the following exercise questions:
 1. Exercise 3.12: scientific
 2. Exercise 3.14: cylinderSurfaceArea
 3. Exercise 3.15: sphereVolume

1.6 Accommodation and Differentiation

In ELL classrooms, you should distribute a handout diagramming the parts of a program with all vocabulary words included on the sample code or screenshot. Turning the pictures on the worksheet into classroom posters will be helpful in helping students remember syntax.

1.7 Video

- BJP 3–4, *Programming with Parameters*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c3-4
- CSE 142, *Scanner* (0:21-23:06)
https://www.youtube.com/watch?v=fo9_kOSS1Y8&start=21

1.8 Forum discussion

[Lesson 3.06 Interactive Programs & Scanner Objects \(TEALS Discourse account required\)](#)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.07 — Pokémon Battle Programming Project

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Write** a program that requests user input and returns data.

1.1.2 Assessments — *Students will...*

- **Write** a program that calculates damage done to Pokémon in a battle.

1.1.3 Homework — *Students will...*

- **Summarize** their class notes since the last exam
 - If they are missing notes, get them from another student or supplement them from the textbook

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Classroom copies** of [WS 3.7](#) LP Battle
- **Video** of sample battle (<http://youtu.be/k7k5lee9xxw?t=48s>)
- **Advanced damage calculator** (<https://pokemonshowdown.com/damagecalc>)

The 8-minute video demonstrates a typical battle sequence from one of the more recent Pokémon versions. If your class does not play the video game, you could show battle footage of the anime series, the card game, or the coin game. As the instructor, you should familiarize yourself with the sequence of a Pokémon battle, so you can help students with procedural decomposition and grade different student solutions.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction to lab & viewing of battle	10min
Student programming practice	40min

1.4 Procedure

The programming project today has students programming a “starter” Pokémon battle sequence. This is a somewhat open-ended assignment, since students can submit a basic program that runs 1 or 2 interactions, or a complete battle sequence, depending on their level of understanding.

Student programs for this assignment will be a lobotomized version of a Pokémon battle since students have not yet learned conditional statements. This is a deliberate move: students can focus on building segments of code that accept basic user input, use the math class to generate random numbers to determine battle outcomes (or roll-of-the-dice or spin for the card-game and coin game versions), and return game text. Capitalize on student frustration by (or motivate students with the prospect of) hinting at a more interactive program after the next few lessons on Ch. 4 and Ch. 5.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction to Lab and Viewing of Battle [10 minutes]

At the beginning of class, introduce the lab and watch the sample battle video.

1.4.3 Student Programming Practice [40 minutes]

1. Have students complete this programming project individually. Before you break out the class for lab time, read the question out loud to the class, taking time to pause between each of the requirements outlined in the lab assignment.
2. Ask students what their very first steps should be.
 - o They should outline their approach in pseudocode or with a structure diagram. Remind them that this documentation should be submitted in order to get full credit for their lab, and refer them to the Algorithm for Solving Problems sheet.
 - o Remind students to tackle one part of the problem at a time. Remind students that it is OK if they leave pseudocode in while they solve a different part of the problem, and partial credit should be given to correct pseudocode.
3. To encourage grit, have students review the steps they should take before raising their hand for a question:
 - o Refer to notes, textbooks, and posters/displayed work around the room.
 - o Work on a different part of the problem if they get stuck, then return to it later.
 - o Ask another student for a hint, tip, or for error-spotting.
4. In an email, on the projector, or as a handout WS 3.7, give student the following questions to work on individually (or, if scaffolding requires it, in pairs).
 - o What is the purpose of the battle?
 - o How many battles are there?
 - o Who are the opponents?
 - o What are the rules of the battle?
 - o What are the objectives?
 - o What are the possible outcomes?
 - o What are the possible moves?
 - o What are the possible results?
 - o What are the possible scores?
 - o What are the possible messages?

1.5.0.1 Emphasize with students...

1.5.0.2 Content - Elements for interface design that is efficient and intuitive for the user

As you begin to receive more input from the user, and generate more output for the user to read, it's important to think carefully about how the user will interact with your program. Sometimes instructions can be unclear for the user and the programmer doesn't realize it.

Be sure to ask specific, detailed questions and be sure to format and display output in a way that the user can easily understand. One of the best ways to determine whether or not your program is user-friendly is to have a classmate, friend or relative use your program. As you watch them use it, take notes about how they interact with the program and about whether or not they are confused about any input our output.

1.5.0.3 PROGRAMMING PROJECT

Complete this programming project using your notes, the text book, and any online or in-class sources you like. Your work must be your own; you may ask a friend to look over your work, or discuss procedural decomposition with you, but you must write all code on your own. To receive full credit on this lab, you must submit a structure diagram or pseudocode-plan for each question.

Recall how to use Scanner to get user input:

```
Scanner console = new Scanner(System.in);
System.out.print("Hello, what is your name? ");
String name = console.nextLine();

System.out.print("What is your age? ");
int age = console.nextInt();
```



1.5.0.3.1 Exercise 1

Write a method called battleStart() that introduces the battle, prompts the user to choose their first Pokémon to battle, and outputs the pairing. battleStart() should also return the name of the Pokemon chosen. Your output should look something like this:

Another trainer is issuing a challenge!

Zebstrika appeared.

Which Pokémon do you choose? Arcanine

You chose Arcanine!

It's a Pokémon battle between Arcanine and Zebstrika! Go!

Call battleStart() from your main() method and store the name of the Pokemon in a variable.

1.5.0.3.2 Exercise 2

Write a method called damage() that takes a Pokemon's name as a parameter and returns the about of HP after damage has been done. damage() should prompt the user for their base stats in order to calculate damage.

Use the following equations for calculating damage:

Modifier = Same Type Attack Bonus (STAB) * Random Damage = Modifier * ((2*Level+10)/250 + (Attack/Defense)*Base + 2)

Hint: The Pokémon game always selects a random number between 0.85 and 1.0.

Your output should look like this:

Zebstrika used Thunderbolt!

Trainer, what are your Arcanine's stats? Level: Attack: Defense: Base: STAB: HP:

Arcanine sustained 10 points damage. HP, after damage, are now 70.

Call damage() from your main() method with the Pokemon's name from Exercise 1 and store the return value (HP) in a variable.

1.5.0.3.3 Exercise 3

Write a method called statsTable() that accepts the user's Pokemon name, stats and learned moves as parameters, and outputs something similar to this image:



You are not required to align the columns of the tables in any fancy way, but if you do, use escape sequences to align data. For your drawing, you may use code you've grabbed from the internet, or recycle an image you created earlier in the year.

Sample output:

Name Alakazam

1.6 Level 40

HP 96 ATTACK 52 DEFENSE 51 SP. ATK 121 SP. DEF 81

1.7 SPEED 107

Moves Learned: Thunder Wave, Hidden Power, Psycho Cut, Recover

Call statsTable() from your main() method with the Pokemon's name from Exercise 1 and the HP from Exercise 2 and any other values you'd like for the other parameters.

1.7.0.0.1 Conclusion

In your completed project should include the following methods:

- - battleStart()
- - damage()
- - statsTable()

These methods should all be called in main() so that the player can experience the entire battle in one sitting.

Proper Java syntax and thorough comments are required.

1.7.0.1 Emphasize with students...

1.7.0.2 Curricular Competencies – Applied Design - Understanding Context - Testing

One of the best ways to effectively design and develop for end-users is to include them in the design and development processes. Computer programmers do this by interviewing the end-users at the beginning of the process. This allows the programmer to understand what the user is looking for and it helps create an initial plan for design ideas.

When it's time for the program to be tested, it is often good practice to have the end-user test the software. This allows them to comment and provide feedback on different parts of the program before it is totally complete. The programmer should take notes related to the end-user's questions and concerns about the program, and about the things that they love.

1.7.0.3 Final Project

Imagine you had to create a program similar to the one above, but instead of using Pokemons it used insects, dinosaurs or basketball teams. What initial stats would each dinosaur have? What attacks would insects have? How would you program the ability to have one basketball team challenge another basketball team?

The final project in this course will provide you with the opportunity to create a program, similar to Pokemon, that uses different characters or "challengers". As you read more about the Pokemon program, think carefully about how you would implement the same ideas in a different context.

1.8 Accommodation and Differentiation

If you have students finish the lab quickly, invite them to check out the advanced damage calculator online. They can add input fields to their own damage calculator, thus improving their Pokémon simulation.

Alternatively, if students seem interested in increasing the interactivity of their battle sequence, you can allow them to read ahead in the book. Students may read up on Bulbapedia (the wiki for Pokémon) that Thunderbolt has a 10% chance of paralyzing its' target. Invite students to think how to add that factor into their battle simulation as they read through Ch. 4 and Ch. 5 materials.

If students are struggling with creating the graphic in Exercise 3, help students by writing helper lines of code on the board, or creating a pseudocode outline as a whole group.

1.9 Forum discussion

Lesson 3.07 Pokémon Battle Programming Project (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.08 – Finding & Fixing Errors

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Find** errors in their returned homework assignments.
- **Correct** their previously submitted homework and classwork

1.1.2 Assessments — *Students will...*

- **Re-submit** all homework assignments with corrected answers.

1.1.3 Homework — *Students will...*

- **Complete** Chapter 4 self-check problems 1–6

1.2 Materials & Prep

- **Any student homework assignments** that you have not yet returned
- **Student self-help system** (such as C2B4 or student pairing)
- **Bookmarks** on students' computers to [webmaker.org](#)

If you are not yet familiar with X-ray Goggles, Thimble, or Popcorn Maker, you should take some time to explore [webmaker.org](#) before class. The site is rich with enrichment tools and hooks for your classroom!

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction and homework distribution	5min
Student work	35min
Students trade work, check, and submit	10min

1.4 Procedure

Today we continue reinforcing concepts and applying the tools, procedures, and code that were introduced last week. Students will have the opportunity to correct any incorrect homework assignments. If students did not have time to finish the programming projects from yesterday, you may allow them time to work on those projects today.

Reward students that did their work correctly with quiet free time. Alternatively, give them a fun programming assignment to do, such as generating a meme, animations, gig posters, or comic strip on Mozilla's webmaker.org. These activities teach HTML and CSS, so you might avoid them if you think exploring different syntax will confuse your students.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Homework Distribution [5 minutes]

1. Return student homework packets, or have students place their returned homework in a pile on their desk.
2. Explain to students that they have the opportunity to get full credit on their homework grades by correcting them now, in class. Ask students for suggestions/ideas on how to make sure they don't miss any errors. (By now students should be used to relying on their error checklist/algorithm.)

1.4.3 Student Work [35 minutes]

Have students work individually to correct their homework grades. - Offer time checks for students so they stay on task. - If students have not finished their programming project from yesterday's class, allow them to do so today.

1.4.4 Students trade work, check, and turn in [10 minutes]

At the end of class, have students trade their homework assignments to evaluate each other's corrections before submission.

1.5 Accommodation and Differentiation

Students that don't have corrections to make should be rewarded for their hard work with silent free time. Encourage them to do work for another class, read the next chapter, or do a fun programming project online.

If you have a student that appreciates public recognition, have them serve as your "TA" this class, going around to help students correct their papers. Remind them to guide students through the process instead of just giving them the answers.

1.6 Forum discussion

Lesson 3.08 Finding & Fixing Errors (TEALS Discourse account required)

formatted by Markdeep_1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.09 – Relational Operators & if/else

1.1 Overview

1.1.1 Objectives – *Students will be able to...*

- **Evaluate** relational expressions
- **Predict and trace** the flow of an if statement.

1.1.2 Assessments – *Students will...*

- **Evaluate** relational expressions and **practice** correct if statement syntax during a game of grudgeball.

1.1.3 Homework – *Students will...*

- **Read** BJP 4.1 “Nested if/else” and “Flow of control”
- **Complete** self-check questions 7–9 and exercises 1 & 2

1.2 Materials & Prep

- **Projector and computer** (optional)
- **Whiteboard and markers**
- **Classroom copies Poster 3.16.2: Operator Precedence**
- **Rules** for grudgeball (see website for details: <http://toengagethemall.blogspot.com/2013/02/grudgeball-review-game-where-kids-attack.html>)

Take the time to familiarize yourself with the rules of grudgeball, and test out your 2 and 3 point lines before class begins (you may need to readjust them). If you can get permission from your school to leave tape on the floor, it is helpful to have those lines down for the rest of the year. In future classes, if your students are having a hard time settling down during a review session, or can't stand a worksheet, you can always convert the worksheet or review session into a quick game of grudgeball.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction & note-taking	15min
Activity: Grudgeball	35min

1.4 Procedure

Rather than drill new rules with worksheets, the drilling/activity portion of the class will serve to tie the lesson together in the form of a class competition. If space and whiteboard setup allow, set up the grudgeball “court” and scoreboard before class begins to excite the students when they walk in.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction & Note-Taking [15 minutes]

Before you begin lecture, announce to students that they should pay close attention, since the lecture content will be tested during the game.

- If you want to write code that executes some of the time, but not all the time, you can write an **if statement** (ask students what situations might use an if statement—if they are stuck, ask them to think about their last few programming projects!)

```
if (test) {           // Boolean Expression
    <statement>;   // Control Statement
    <statement>;   // Control Statement
}
```

- Have students come up to the front of the room to demonstrate the flow of control of the if control structure.
- Put 2 alternatives together by using an if/else statement:

```
if (test) {
    <statement>;
    <statement>;
} else {
    <statement>;
    <statement>;
}
```

- So what should students put in the “test” section to evaluate as true or false?

```
3 * 3             ==          3 * 1 * 3
// <expression> <relational operator=""> <expression>
```

Evaluate both expressions, then see if the relational operator holds true or not.

- Operators students need to know:

- ==: equal to
- !=: not equal to
- < : less than
- > : greater than
- <=: less than or equal to
- >=: greater than or equal to |

- As far as precedence goes:
 - relational operators have a lower precedence than arithmetic operators
 - relational operators have higher than equality operators
 - inequality operators ($<$, $>$, \leq , \geq) have higher precedence than the equality operators ($==$, $!=$)
- $3 + 2 * 2 == 9$ evaluates to: false, since 7 is not equal to 9.

Have students direct you how to solve this.

If you'd like to refer to a visual aid for this segment of the introduction, poster 3.16.2 illustrates Java rules of precedence for all operators.

- You can only use a relational operator on primitive data types! (Ask students which types are included, which ones are excluded.)
- Review the mod % operator. Ask the students how they could use the mod operator to check for if a number is an odd or even number. Follow up with can a similar algorithm be used to check if any number is evenly divisible by another.

1.4.3 Activity: Grudgeball [35 minutes] [Optional]

If you feel like your class has a grasp on the syntax and relational expressions, consider skipping this game and focusing on on-the-board examples (you can use the questions from Grudgeball below) or moving on to 3.10.

1. Divide students into their assigned teams.
 2. Review the rules for grudgeball, and have the students repeat the rules back to you.
 3. Using the problems listed below (and any you may add, depending on your class' needs), play grudgeball until a team wins, or until the class period ends.
 1. If a class gets the answer wrong, BRIEFLY pause the game to have students offer corrections before moving to the next team's question.
 2. If correction seems to be dragging on, jump in and quickly re-teach using the incorrect answer as your example. It is important to keep the pace going to maintain student interest in the game!
 4. Grudgeball problems & answers have been grouped assuming that you have 6 teams. If you have fewer teams, each "round" will be shifted accordingly, so you may have rounds where different teams are practicing different concepts. Judge each team's knowledge gaps, and adjust which questions you ask each group accordingly.
-

1.4.3.1 GRUDGEBALL PROBLEMS

Translate these statements into logical tests that could be used in an if/else statement. Write the appropriate if statement with your logical test.

- a) z is odd.
- b) z is not greater than y's square root.
- c) y is positive.
- d) Either x or y is even, and the other is odd.
- e) y is a multiple of z.
- f) y is a non-negative one digit number.

Given the variable declarations

```
int x = 4;
int y = -3;
int z = 4;
```

what are the results of the following relational expressions? (True or False?)



- g) $x == 4$
- h) $x == y$
- i) $x == z$
- j) $x + y > 0$
- k) $y * y <= z$
- l) $x * (y + 2) > y - (y + z) * 2$

Correct the following statement syntax errors:

- m) if $x = 10$ then {
- n) if [$x = 10$] {
- o) if (x equals 42){
- p) if ($x == y$){
- q) If ($x > 8$){
- r) IF($x <= 7$){

Identify and correct one of the (7) errors in the following code:

```
public class Oops4 {

    public static void main (String[] args) {
        int a = 7, b = 42;
        minimum(a,b);
        if {smaller = a} {
            System.out.println("a is the smallest!");
        }
    }

    // Returns the minimum of the parameters a and b.
    public static void minimum (int a, int b) {
        if (a < b) {
            int smaller = a;
        } else (a = b) {
            int smaller = b;
        }
        return int smaller;
    }
}
```

//

1.5 College Board Topic Questions

After this lesson, students will be able to answer questions from the College Board Unit 3 Topic Questions 3.1 Boolean Expressions, 3.2 if Statements and Control Flow, 3.3 if-else statements, and 3.4 else if Statements.

1.6 Accommodation and Differentiation

In ELL classrooms, you should read each question aloud in addition to showing it on the board or projector.

If this review session is too easy, give students time to start on the homework once you have finished Grudgeball.

1.7 Teacher Prior CS Knowledge

- The not operator ! (commonly read as “bang”) is commonly used in logical expressions to negate a boolean value. The use of not is unique among the logical operators as it takes only one operand. Not is comparable to the negative sign (-) where a minus before a numeric value or expression represents the negative of the value whereas a not (!) before a boolean value represents the opposite truth value: true becomes false and false becomes true.
- Programming languages have a default order of evaluating expressions commonly referred to as operator precedence. We usually teach students to be explicit in their logical expressions for clarity and to avoid

mistakes. However, a knowledge of the language's precedence rules is valuable when both reading and debugging code. See the following for Java's precedence rules:

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html>. In order to grade correctly, it is essential to know Java's order of operations.

1.8 Common Mistakes

Conditionals common mistakes: <http://interactivepython.org/runestone/static/javareview/conditionals/cmistakes.html>

1.9 Misconceptions

- Confusing the assignment operator (=) with the comparison operator (==) is one of the most common mistakes done by beginning programmers. A single equal sign is used to denote equality in mathematics and for most students they have been doing math much longer than computer science.

It is important for the teacher when reading code aloud to differentiate single equals as an assignment and double equals as a comparison. The following would be read as:

```
x = 5;      // "x is assigned 5"  
if (y == 4) // "if y is equal to 4"
```

- Many students think all statements in Java end in a semi-colon (;). However, this is not always the case as the if-block ends in curly braces when used with a block of code. Students in the habit of adding semi-colons to the end of each statement inevitably add semi-colons at the end of the conditional, like so:

```
if (a > 1); // all statements end in semi-colon misconception
```

When teaching students about Java statements, it is important to distinguish between statements that end in a semi-colon and those that signify scope with the curly-braces. Single Java statements such as declaring variables, assignment, etc. end in semi-colons while Java statements that denote scope like class definitions and methods have curly braces. If-statements can do both.

- Logical operators AND and OR are covered in a future lesson. However, students will combine logical operators similar to math syntax:

```
if (9 <= grade <= 12) // relational operator misconception
```

For students that ask about compound conditionals, you will need to defer and state they will be covered shortly in a future lesson. For students with prior programming experience, they will search for the correct syntax because the English words "and" and "or" do not work in Java.

- One common error is mismatched the parentheses:

```
if (a == 10 // mismatched parentheses
```

or mismatched curly braces:

```
if (b == 12)  
    dozen = true;  
}
```

Even though many IDEs will help students by providing matching parentheses, a good habit for students to learn when writing code by hand is to add the closing parentheses right after they write the open parentheses. The same is true for open and close curly braces. Students will need to learn to write code by hand for the AP exam, so practice hand written Java is important to students' success.

In addition, good coding style with proper indenting helps with student recognition of mismatched closing parentheses and curly braces because of the visual pattern. Good programming style makes it easier to identify errors in the pattern. This can be used during lab for students with mismatched parentheses and/or curly braces. Instead of find the error, asking the student to clean up their indenting will help them find errors on their own.

- When constructing if-statements, students insert the condition of the if-statement within the brackets instead of the parentheses:

```
if {a > 1} // conditional vs block misconception
```

//

See above "mismatched the parentheses" for good programming practice.

1.10 Video

- BJP 4-2, *Nested if/else Statements* (note: title of video is mislabeled, should be "if/else Statements")
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c4-2
- CSE 142, *if/else Statements* (38:41-50:15)
https://www.youtube.com/watch?v=fo9_kOSs1Y8&start=2321
- CSE 142, *Methods with Conditional Execution* (4:08-18:39)
<https://www.youtube.com/watch?v=omqNzejWfSY&start=248>
- CS Homework Bytes, *Relational and Conditional Operators, with Elizabeth*
<https://www.youtube.com/watch?v=M-mYpnPygYo>
- CS Homework Bytes, *If-Statements, with Jim*
<https://www.youtube.com/watch?v=SxWFYfA4ioM>

1.11 Forum discussion

Lesson 3.09 Relational Operators & if/else (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.11 – Reducing Redundancy

1.1 Overview

1.1.1 Objectives – *Students will be able to...*

- **Simplify code and reduce redundancy** by factoring if/else statements and testing multiple conditions simultaneously.

1.1.2 Assessments – *Students will...*

- **Complete** a class competition

1.1.3 Homework – *Students will...*

- **Read** BJP 4.2
- **Finish outlining** Chapter 4, excluding sections 4.3, 4.4, and 4.5

1.2 Materials & Prep

- **Projector and computer** (optional)
- **Whiteboard and markers**
- **Rosters** for class teams

The teams for today's competition should be tiered. Depending on the size of your class, you should aim for 4 teams or teams of 4 people.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Reviewing procedures and forming groups	5min
Competition question 1	15min
Competition question 2	15min
Students begin outlining Chapter 4	15min

1.4 Procedure

As your hook, grandly announce a class competition between teams and announce the prize for the winning team (this might be TEALS swag, bonus classroom participation points, or additional raffle entries in the year-end TEALS giveaway). Use this class competition as an assessment to determine how much re-teaching you need to do.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Reviewing Procedures and Forming Groups [5 minutes]

Announce class teams and rearrange students as needed.

1.4.3 Competition Question 1 [15 minutes]

Give students 5 minutes to complete the challenge, and take note of which team finishes first.

- If students are struggling, you may extend the time, or offer universally helpful tips (e.g. "If you see parentheses that aren't being used to establish precedence, or to cast, it means you're calling another method.")

The team that has the correct answer first wins the prize.

- Review student answers together as a whole group, revisiting concepts taught earlier in the week as mistakes come up. Whenever possible, have students volunteer the correct procedure, approach, or code. Encourage students to take notes during this process so they can review topics outside of class.
-

1.4.3.1 COMPETITION QUESTION 1

Factor out redundant code from the following example by moving it out of the if/else statement, preserving the same output.

```
if (x < 30) {
    a = 2;
    x++;
    System.out.println("Spongebob Squarepants! " + x);
} else {
    a = 2;
```

```
        System.out.println("Spongebob Squarepants! " + x);
    }
```



1.4.4 Competition Question 2 [15 minutes]

Once you've completed this exercise, offer up the second challenge question, as before. This question may take up to 10 minutes for students to complete.

1.4.4.1 COMPETITION QUESTION 2

Rewrite the poorly-structured code given here, so it avoids redundancy. For the case of this competition, you can assume that the user always inputs 1 or 2.

```
int sum = 1000;
Scanner console = new Scanner(System.in);
System.out.print("Is your money multiplied 1 or 2 times? ");
int times = console.nextInt();
if (times == 1) {
    System.out.print("And how much are you contributing? ");
    int donation = console.nextInt();
    sum = sum + donation;
    count1++;
    total = total + donation;
}
if (times == 2) {
    System.out.print("And how much are you contributing? ");
    int donation = console.nextInt();
    sum = sum + 2 * donation;
    count2++;
    total = total + donation;
}
```



1.4.5 Students Begin Outlining Chapter 4 [15 minutes]

Once you have completed the competition and review, have students begin outlining Chapter 4. Whatever they do not finish outlining they should complete tonight for homework.

1.5 Accommodation and Differentiation

Circle around the room to help students through reading the text in the textbook. Make sure that each of your working teams are properly stratified (rather than using tiered grouping). If you teach in an ELL classroom, you may opt to change the assignment so that all the lines of code are present, but shuffled out of order (as a Parsons-type Problem).

If students are speeding along, encourage students to write down questions to pose to other groups during mini-lessons.

1.6 Video

- BJP 4-3: *Factoring if/else*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c4-3
- CSE 142: *Factoring if/else Statements and Reasoning about Paths* (18:40-34:05)
<https://www.youtube.com/watch?v=omqNzejWfSY&start=1120>

1.7 Forum discussion

Lesson 3.11 Reducing Redundancy (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.13 – while Loops

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Trace** while loops to predict:
 - The number of times the body executes
 - The output of the code
- **Differentiate** between while loops, if statements, and for loops

1.1.2 Assessments — *Students will...*

- **Complete** Practice questions

1.1.3 Homework — *Students will...*

- **Read** BJP 5.1 (skip “do/while Loops”)
- **Complete** self-check questions \#1-4 and exercise 2

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Classroom copies** of [WS 3.13](#)

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction & think-pair-share	15min
Student practice activity	35min

1.4 Procedure

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction & Think-Pair-Share [15 minutes]

1. Ask students to offer pseudocode that explains how they might track damage to a Pokemon.
 - (Student answers should include some of these steps: Start with initial HP, and while HP is greater than 0, ask how much damage, subtract it, and end print with “Pokemon fainted!”) Point out that this process has no predetermined length (indefinite looping), so you need to use a new type of loop called a while loop.
 - If students need additional examples of indefinite looping, use a simpler example, asking students how they would double a number until it was bigger than N.
2. Engage students in the introduction today by having students complete the graphic organizer on WS 3.13 as you review the structure, flow, and syntax of the while loop.
3. Compare and contrast the while and for loops (see code snippets below): both are control structures that send the flow of control through a loop, but scope differs, so the loops execute in different ways.
 - Have students point out where i is declared.
 - Introduce the concept of definite vs. indefinite loops and ask students when they might want to use an indefinite loop (they will probably have wanted to use this structure in their earlier programming projects—prompt them with this if no one volunteers an example).
 - Call 2 students up to the board; one to trace the flow of control and the other write the output.

```
// while loop:                                // for loop:  
int i = 0;                                     for (int i = 0; i < 10; i++) {  
while (i < 10) {                               System.out.println (i);  
    System.out.println (i);                      }  
    i++;  
}
```

4. Introduce tracing while loops by using a trace table:

- Columns represent variables
- Rows represent values during each iteration

```
5. int n = 172;  
    int sum = n;  
    while (n<0) {  
        int digit = n % 10;  
        sum = sum + digit;  
        n = n / 10;  
    }  
System.out.println (sum);
```

- Trace the above code using the trace table below.

6.

n	sum	digit	output

5. Invite students to Think-pair-share on the following example:

```
int n = 91;  
int factor = 2;
```

```
//  
while (n % factor != 0) {  
    factor++;  
}  
System.out.println("First factor is " + factor);
```

- How many times does this loop execute? What is the output?
- In the same pairs, have students rewrite the while loop as a for loop.

1.4.3 Student Practice Activity [35 minutes]

1. Have students complete the following practice problems:
 1. Self-Check 5.1: whileLoops
 2. Self-Check 5.2: forToWhile
 3. Self-Check 5.3: whileLoopMystery1
 4. Self-Check 5.4: whileLoopMystery2
2. If students complete these problems with time to spare, have them complete Exercise 5.2: gcd.

1.5 College Board Topic Questions

After this lesson, students will be able to answer questions from the College Board Unit 4 Topic Questions 4.1 While Loops and 4.5 Informal Code Analysis.

1.6 Accommodation and Differentiation

If you have students that finished the classwork ahead of time, encourage them to explore do/while loops

1.7 Video

- CSE 142, *While Loop* (11:21-15:55)
<https://www.youtube.com/watch?v=hpDQ9tdrj1Y&start=681>

1.8 Forum discussion

[Lesson 3.13 while Loops \(TEALS Discourse account required\)](#)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.15 — Fencepost & Sentinel Loops

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Describe** when to use fencepost and sentinel loops.
- **Use** proper syntax to construct these control structures.

1.1.2 Assessments — *Students will...*

- **Teach** a mini-lesson explaining the relationship between parameters and values stored in memory

1.1.3 Homework — *Students will...*

- **Read** BJP 5.2
- **Complete** exercises \#6 & 8
- **Summarize** all of your daily notes if not already done

1.2 Materials & Prep

Day 1 - **Projector and computer** - **Whiteboard and markers** - **Group copies** of [WS 3.15](#) - **3 or more classroom copies** of the textbook

Day 2 - Teacher access to CS Awesome [**Unit 4 Lesson 03 Loops and Strings Lesson Plan**] Sign up at [CS Awesome AP CSA Java Curriculum] - Access to CS Awesome [**4.3. Loops and Strings**]

1.3 Pacing Guide Day 1

Section	Total Time
Bell-work and attendance	5min
Mini-lesson planning & prep	15min
Student presentations & practice	30min

1.4 Pacing Guide Day 2

Section	Total Time
Bell-work and attendance	5min
CS Awesome 4.3 Loops and Strings	30-40min
Wrap up	5min

1.5 Procedure Day 1

Today 3 student teams will teach a lesson on fencepost algorithms, sentinel loops, or sentinel loops with if statements. Your hook will be to turn the class over to students as soon as they enter. Student groups are expected to generate sample questions; let students know that you will collect those questions to use on quizzes or as bellwork.

1.5.1 Bell-work and Attendance [5 minutes]

On the board, on the projector, or in a group handout, let students know that they need to prepare a 5 minute lesson and a 2- 5 minute class activity to teach their topic. Use the grading rubric as outlined here:

3 pts.	2 pts.	1 pts.	0 pts.
Presentation includes definitions and an example with proper syntax.	Presentation includes definitions or an example with proper syntax.	Presentation includes definitions or an example with proper syntax with one mistake.	Presentation includes definitions or an example with proper syntax with many mistakes.
Presentation includes a non-example as helpful contrast.	Presentation includes a non-example that is marginally helpful.	Presentation includes a non-example that does not add to comprehension.	Presentation includes a non-example that adds confusion, or presentation does not include a non-example.
Presentation includes a helpful tip that is clearly explained and concisely stated.	Presentation includes a helpful tip that is clearly explained or concisely stated.	Presentation includes a helpful tip that is not clearly explained and may include a small error.	Presentation does not include a helpful tip or hint.

1.5.2 Mini-Lesson Planning & Prep [15 minutes]

1. Assign each group a subsection of section 5.2 “Fencepost Algorithms,” and make sure that you circle that assignment on each groups’ copy of WWS 3.15. Student groups should take 15 minutes to review their section, re-read the example on the pages following the example, then figure out how they want to explain the algorithm to the class.
2. On the board or overhead, give students a few things they should consider in planning their mini lesson:
 - a. Who is going to speak when?
 - b. How are you going to illustrate the flow of control?
 - c. What do you need to have up on the board to illustrate your mini-lesson, and who is in charge of writing it out?
 - d. Where and how will you feature the output produced by your code segment?
 - e. What is your mini-activity going to look like? (You might want to assign 1–2 people to work on this section while the rest of the group works on the lesson.)
3. Have student groups sequentially teach through fencepost algorithms, sentinel loops, and fenceposts with if statements.
4. Encourage students to add these strategies to their Tricky Code Cheat Sheet.

1.5.3 Student Presentations & Practice [30 minutes]

1.6 Accommodation and Differentiation

If your class learns better through tactile or visio-spatial learning, you can change this assignment to a make-a-poster lesson, having students work in pairs or triplets to create an informative poster on one of the topics. If you’re fortunate enough to have a theatrical or musical class, invite them to create a song, poem, or narrated dance/play that teaches their topic. For poems or songs, encourage students to write them out or record them so you can display them around your room.

1.7 Procedure Day 2

This lesson continues with fencepost and sentinel loops as applied to Strings. It utilizes the CS Awesome's Lesson 4.3 Loops and Strings.

1.7.1 Bell-work and Attendance \[5 minutes\]

1.7.2 Loops and Strings \[35 minutes\]

The student lesson for Part 2 uses CS Awesomes [4.3. Loops and Strings]. There you will find the lesson plan and activities to check for student understanding. The teacher lesson plans are accessible by joining the [Teaching CSAwesome google group] and navigating to [Unit 4 Lesson 03 Loops and Strings Lesson Plan]. Have students log into [4.3. Loops and Strings] and complete activitie 1-4.

1.8 Common Student Questions

Since your student-instructors won’t be able to answer in-depth questions on their topic, you should be ready to assist during the Q & A section of their lesson. Some student questions that have popped up in the past, with their answers, are listed below:

1. The sentinel loop example in the book says that the sentinel value will be read and added to the sum unless we do a fencepost algorithm or if statement. Why is that? Doesn't the test evaluate to false and terminate the loop?

Write an example on the board in pseudocode or actual code, and trace the flow of control with your marker. In this case, the pseudocode in the book is a bit misleading because it looks like the test will terminate in the header. The prompt for the sentinel is already in the loop body, so the sentinel will be evaluated before loop termination.

If you think your class will be comfortable, instruct your teaching group to do this illustration, and just help along.

2. What do you do if you don't know the first value to put in your fencepost algorithm? What if you're getting all of your data from user input?
3. When do we know to reverse the order of loop construction? The example from the book has us switching around a lot of stuff for the sentinel loop with if statements.

Put an example up on the board (or have the student instructors do so), and trace the flow of control before and after rearranging the loop body.

1.9 Video

- BJP 5-3, *Sentinel Loops*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c5-3
- CSE 142, *Fencepost* (0:28-11:20)
<https://www.youtube.com/watch?v=hpDQ9tdrj1Y&start=28>
- CSE 142, *Sentinel Loops* (15:56)
<https://www.youtube.com/watch?v=hpDQ9tdrj1Y&start=681>

1.10 Forum discussion

[Lesson 3.15 Fencepost & Sentinel Loops \(TEALS Discourse account required\)](#)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.17 — Finding & Fixing Errors

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Find** errors in their returned homework assignments.
- **Correct** their code

1.1.2 Assessments — *Students will...*

- **Re-submit** all homework assignments with corrected answers.

1.1.3 Homework — *Students will...*

- **Study** for the test by:
 - **Reviewing** all the blue pages at the end of Chapters 3, 4, and 5
 - **Re-reading** sections as needed
- **Submit** 5 questions for review in class tomorrow using electronic survey

1.2 Materials & Prep

- **Any student homework assignments** that you have not yet returned
- **Student self-help system** (such as C2B4 or student pairing)
- **Electronic survey** for student review requests

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction and homework distribution	5min
Student work	35min
Students trade work, check, and submit	10min

1.4 Procedure

Today, students will have the opportunity to correct any incorrect homework assignments. If students did not have time to finish the programming projects from yesterday, you may allow them time to work on those projects today.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Homework Distribution [5 minutes]

1. Return student homework packets, or have students place their returned homeworks in a pile on their desk.
2. Explain to students that they have the opportunity to get full credit on their homework grades by correcting them now, in class. Ask students for suggestions/ideas on how to make sure they don't miss any errors. (By now students should be used to relying on their error checklist/algorithm.)

1.4.3 Student Work [35 minutes]

Have students work individually to correct their homework grades. - Offer time checks for students so they stay on task. - If students have not finished their programming project from yesterday's class, allow them to do so today.

1.4.4 Students trade work, check, and turn in [10 minutes]

At the end of class, have students trade their homework assignments to evaluate each other's corrections before submission.

1.5 Accommodation and Differentiation

If you have students who are speeding through this lesson, you should encourage them to tackle programming project 1 in Chapter 5.

If you were unable to finish grading student notebooks yesterday, finish them today while students are working. Return notebooks by the end of class so students may use them to study for the exam.

1.6 Forum discussion

Lesson 3.17 Finding & Fixing Errors (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.18 – Consumer Review Lab

1.1 College Board 2019 Labs

In the summer of 2019, the college board released 4 new labs. Either FracCalc or the Consumer Review Lab can be assigned after the completion of Unit 3.

1.2 Overview

1.2.1 Objectives — *Students will be able to...*

- **Complete** a long-form lab, using string literals, static methods, if statements, while loops, algorithms, and the String class

1.2.2 Assessments — *Students will...*

- **Complete** College Board's AP CS A Consumer Review Lab
- **Answer** end of activity Check your understanding and complete Open-ended activity.

1.2.3 Homework — *Students will...*

- **Complete**

1.3 Materials & Prep

- **Projector and computer**
- **Consumer Review Lab Teacher's Guide**
- **Classroom copies** of the Consumer Review Lab Student Guide
- **Associated Consumer Review Files**

Read through the Teacher and Student guides ahead of time to familiarize yourself with the parts of this long-form lab. Using the guides, complete the lab on your own to spot possible challenges for your students. Upload all student files onto each computer desktop for student access.

1.4 Pacing Guide: Day 1

Section	Total Time
Student Activity 1	Full class - students
Notebook checks	Full class - teacher
Homework:	

1.5 Pacing Guide: Day 2

Section	Total Time
Student Activity 2	Full class - students
Notebook checks	Full class - teacher
Homework:	

1.6 Pacing Guide: Day 3

Section	Total Time
Student Activity 3	Full class - students
Notebook checks (if not completed)	Full class - teacher
Homework:	

1.7 Pacing Guide: Day 4

Section	Total Time
Student Activity 4	Full class - students
Notebook checks (if not completed)	Full class - teacher
Homework:	

1.8 Pacing Guide: Day 5

Section	Total Time
Student Activity 5	Full class - students
	Full class - teacher
Homework:	

1.9 Pacing Guide: Day 6

Section	Total Time
Student Activity 5 (day 2)	Full class - students
	Full class - teacher
Homework:	

1.10 Procedure

All guides, sample code, answer code, and example code may be found by logging into the College Board AP Audit side and downloading the Consumer Review Lab materials.

1.10.1 General Project Notes

- Open IDE and guide students through opening the Consumer Review files.
- Encourage students to use their Tricky Code Cheat Sheets, 4 Commandments of Scope, notebooks, textbooks, classroom posters, and homework assignments.
- Offer occasional time-checks to help keep students on pace.
- Grade notebooks and review books in between helping students so students can keep notebooks for homework and studying in the evenings.

1.11 Accommodation and Differentiation

In ELL classrooms, read all directions aloud before breaking into individual practice, and allow up to twice the amount of time for completion of the lab. As needed, allow students to pair up to help each other with reading comprehension (but remind students that they each must submit their own code). Each day that you begin the lab, start with a quick survey of student concerns and questions.

1.12 Forum discussion

Lesson 3.18 Consumer Review (TEALS Discourse account required)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.19 — Review

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Identify** weaknesses in their Unit 1 knowledge.

1.1.2 Assessments — *Students will...*

- **Create** a personalized list of review topics to guide tonight's study session.

1.1.3 Homework — *Students will...*

- **Study** for tomorrow's test using your targeted review list

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and marker**
- **Results** from electronic survey of review topics
- **Classroom copies** of the practice test [WS 3.18](#)

Once students have submitted their review requests, assemble those topics into categories and prepare to re-teach the topics as needed.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction and test format orientation	15min
Test review	30min
Check student study lists	5min

1.4 Procedure

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Test Format Orientation [15 minutes]

1. Clearly indicate that you expect all students to have a list of review topics to study this evening. Periodically remind students that this list will be checked at the end of class.
2. Students should already be familiar with the sections of the test, but it doesn't hurt to have students re-read the directions.
3. Work through the sample problems on the test as a way of reviewing topics, and answer any questions that students bring up as you go.

1.4.3 Test Review [30 minutes]

1. Work through the various review topics, prioritizing questions that popped up the most.
 - a. Some questions you may already have addressed while working through the sample test.
 - b. Be ready for additional questions to pop up as you go. Save yourself the work and use old homework questions and student-generated test questions as examples to work through.
2. Use a combination of group-solving questions on the whiteboard, think-pair-share, and timed-response as review strategies.
3. After you've completed reviewing an idea, remind the class that they should write down that topic if they feel they still have to review it tonight. (Yes, this will be a reminder every few minutes, but it will pay off later when students start creating review lists without prompting later in the year!)

1.4.4 Check student study lists [5 minutes]

Spend the last 5 minutes of class checking each student's review topic list.

1.5 Accommodation and Differentiation

The first practice problem calls a method inside an expression inside a parameter to answer another method call (this sounds crazy, but take a look at the question before you write it off!) Logically, the question makes

sense, but it may throw some of your students. Use the question as an opportunity to model proper test-taking strategies:

- Read code line-by-line.
- If stumped on a multiple-choice question, try plugging in the answers to see if they evaluate correctly.
- Write notes and cross out answers on your paper copy of the test.

If you have been using Parsons Problems, on your students' tests you may want to throw in a "full challenge" blank section 2 question during this unit or the next to scaffold your students up to the challenge of a real AP test.

As written, the exams increase in length and complexity with each unit. If your students are all acing the test, challenge your students by modifying the section 2 questions, and adding extra section 1 questions.

1.6 Forum discussion

[Lesson 3.19 Review \(TEALS Discourse account required\)](#)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 3.XX1 – Programming Project (FracCalc Alternative)

1.1 ## Overview

1.1.1 Objectives — *Students will be able to...*

- **Conduct user-centred research** to identify specific functions for a specialized calculator application
 - **Plan and create** a calculator that perform specialized operations for an end-user
 - **Test, evaluate, and share** the end product
-

1.1.1.1 Emphasize with students...

1.1.1.2 Content - arithmetic operations, data representation, applied design

The “brain” of the computer is a block of circuitry called the “Central Processing Unit” (CPU). Inside the CPU, there is a sub-block of circuitry called the “Arithmetic Logic Unit” (ALU). As you might guess, this block performs arithmetic operations!

At a most basic level, the computer is simply, a “deluxe calculator”. While hand-held calculators have limitations on the size of numbers you can work with, your desktop or laptop computer can support much larger numbers. Not only can the computer calculate numbers quickly, and accurately, it is also great at doing it over, and over, and over again. The computer never gets bored.

In what situations would you need to perform lots of calculations, possibly on very big, or very small, numbers? How often do you still use, or when would you use, a hand-held calculator (or the calculator on your phone)?

1.1.2 Assessments — *Students will...*

- **Apply** advanced data and control concepts covered in Unit 3
- **Submit** a complete, functional program

1.1.3 Pacing Recommendation

- This project is intended to be the same length as the FracCalc Project.
- The duration of this project is at the discretion of the teacher. About 4-5 classes are recommended.
- Project involves conducting research work (survey or interviews), and communicating with end-user, outside of the classroom.

1.2 Implementation Details

1.2.1 Complexity and Creativity

This project is an alternative to the FracCalc project in the existing AP Computer Science course. Students should come up with the idea themselves, based on user-centred research, and ideate a calculator application to address the needs of a specific user group. The “calculator application” could possibly involve several steps, such as solving the quadratic formula, or sharing the cost of a party.

Students will follow applied design process to implement the idea. You should talk to your teacher often to ensure that your progress is in-line with expectations.

1.2.2 Documentation and Style

As with all projects, your program must be well-written, well-documented, and readable. Writing code with good style is always good idea. This will help you debug, pick up where you left off each day, and keep track of progress.

STEP 1 - UNDERSTANDING CONTEXT

Conduct user-centred research to find design opportunities and barriers.

Select an end-user for whom you will design and create this program (this can be a friend, classmate, relative, etc). Create interview questions that will allow you to understand the end-user’s interests and likes/dislikes. Since we are creating a calculator application, here are some possible questions:

- when was the last time you used a calculator?
- what do you use it for?
- what are your most common uses for the calculator?

As the interview progresses, you may prompt them with ideas, but also give them time to think. Some possible uses for the calculator may be: calculate how to share costs for meal; calculate the cost of something with a special discount (eg, “buy one, and get one half price”); or calculate cost of something when travelling in foreign country (and compare with the cost of same item back home!).

You may also ask the user about more complex problems that requires a formula (or several steps) to solve. Examples:

- solving the quadratic formula
- factoring polynomials
- cost sharing (eg: 5 people all bought something for the party, how much does each person “owe”, or “gets paid back”)
- difference in cost for filling up gas in the US vs in Canada (involves metric/imperial units, and currency conversion)

At this point, you will need to ask more specific questions:

- “what type of conversions do you do the most?”
- “give me an example of some calculation related to [something they mentioned]”
- “what would really be handy in this situation?”

1.3

STEP 2 - DEFINING AND IDEATING

Choose a design opportunity and point of view, make inferences about limitations and boundaries. Take creative risks to identify gaps to explore, generate a range of possibilities, prioritize ideas for prototyping.

Using the responses from your end-user interview, begin to develop a plan for your custom calculator. Given the duration of the project, you should limit your calculator to do *an interesting calculation* for a *specific end-user*. At this point you will only be be to implement a text-based user interface (ie, no graphical elements, like buttons or scrollbars). Keep it simple and easy to use.

List:

- * the types of calculator functions it will do:
 - what will the user input be?
 - what output will be calculated?
 - what are example data for testing?
- * what user interaction with the program will the user have?
 - a message to prompt for input?
 - a mesage to report on the caluculated output?
 - type a word ("quit") to exit?

Share these ideas with your end user. Record their comments, suggestions and feedback and note any changes that you may make as a result of this interview.

Identify any issues or problems that might arise as you begin to program (what areas might require more information or programming solutions? Where can you find this information?)

CHECKPOINT 1

SUBMIT YOUR INTERVIEW QUESTIONS, RESPONSES AND PROGRAM DESIGN PLAN TO YOUR TEACHER.

STEP 3 – PROTOTYPING AND TESTING

Construct prototypes, making changes to code as needed.

Program your calculator. Be sure to review course notes and activities to make sure that you effectively implement object oriented programming design for your application.

When you are ready, create a short test plan and test your program. Include expected output and actual output and include details related to any fixes that needed to be made. A good motto to remember is **code a little, test a little**

Have your end-user try a working version of your program. Note their suggestions, comments and feedback. Indicate any changes that you made to the program, based on the user's feedback.

CHECKPOINT 2

SUBMIT THE CURRENT VERSION OF YOUR PROGAM TO YOUR TEACHER, AS WELL AS YOUR TEST PLAN AND FEEDBACK FROM THE END-USER

STEP 4 – SHARING, TESTING AND FINAL ITERATION

Gather feedback from users over time to critically evaluate your design and make changes to product design or processes Identify new design issues

Share your work with other classmates, friends or family. Record any feedback, suggestions and comments and use this information to make the final iteration of your program.

CHECKPOINT 3

SUBMIT THE FINAL VERSION OF YOUR PROGAM TO YOUR TEACHER, AS WELL AS YOUR FEEDBACK FROM CLASSMATES, FRIENDS OR FAMILY.

Grading Scheme/Rubric

Implementation	
Project is appropriately complex and creative	4 points
Program is well-documented and shows good style	10 points
Final product meets all requirements and goals laid out in checkpoint specifications	8 points
Program uses programming concepts effectively, including all required elements with an appropriate level of complexity	4 points
Object Oriented Programming concepts are effectively applied with an appropriate level of complexity	6 points

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 4.00 — Test Review & Reteach

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Re-learn or strengthen** content knowledge and skills from Unit 3.

1.1.2 Assessments — *Students will...*

- **Re-submit** test answers with updated corrections for partial or full credit
 - Credit depends on instructor preference

1.1.3 Homework — *Students will...*

- **Read** BJP 6.1
- **Correct** any incorrect test answers by re-answering on a separate sheet of paper
 - To get back credit, they must justify their new answers
 - Staple new answer sheet to old test and turn in tomorrow

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Corrected student tests**
- **Student grades** (posted online, emailed to students, or handed back on paper in class)
- **Digital copy of test questions** for projector

If your class is acing the exams, and you do not feel that you need to re-teach any material, you should skip this lesson and move on directly to LP 4.1. Be sure to update student homework so students have read §6.1.in time for the lesson.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Class discussion (if needed)	10min
Test review and reteach	35min
Check student notes and return tests	5min

1.4 Procedure

Return student grades before class begins or while students are completing the bellwork.

Do not return students' tests before the review session, since you want to motivate students to pay attention to the entire review, taking supplemental notes the entire time.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Class Discussion (if needed) [10 minutes]

1. If grades are low, invite the class to a discussion of what can be improved. Begin with student complaints and suggestions to build student buy-in. Ask students:
 - How they felt they were going to do before the test
 - What surprised them once they were taking the test
 - What they felt worked in the first unit (lessons, review strategies, assignments)
 - What do they think they want to change for the second unit
2. Once you feel that a dialogue has been established, validate students' feelings, then challenge them (e.g. AP courses are stressful, but this is good practice for college, where the pace is faster and professors don't give personalized instruction). In a non-judgmental, supportive tone, remind students that to be successful in the course:
 - Reading is mandatory
 - Homework is mandatory (And valuable! You will never assign "busy" work.)
 - To better manage their time, students should plan for 1 hour of homework a weeknight, with up to 2 hours of homework each weekend. If this seems impossible, they should meet with you or their guidance counselor to assess whether they can fit in an AP class at this time.
 - It is VERY important to keep your tone sympathetic at this point—an overworked, overstressed, underperforming student will slow your entire class down, and color that student against CS for the future!

1.4.3 Test Review and Reteach [30 minutes]

1. Walk the students through each question on the test, glossing over questions that everyone answered correctly.
 - a. You can ask for students to volunteer answers, or call on students randomly. Make sure that students explain their logic when they answer. If a student gives an incorrect answer, the explanation will tell you what you need to re-teach or clarify.

- b. Do not skip questions that everyone answered correctly, but do not spend more than the time it takes to read the question, and congratulate students' correct answers.
2. Project a copy of each question as you review—this will help students recall the question/process the information.
3. Make sure that students are taking notes during the re-teach, reminding students that for homework, they will have an opportunity to win back some of the points on their exam.
4. For Section II questions, select a sample of student work (with any identifying information obscured), and work through the answer together as a class.

1.4.4 Check student notes and return tests [5 minutes]

At the end of class, check student notes, and return the tests in hard copy form if applicable.

1.5 Accommodation and Differentiation

If students' grades are suffering because the reading assignments are taking them too long, you have a few options (some more drastic than others):

- Set aside classroom time to read through the assignment before students leave.
- Give students the lines of code needed to complete assignments, but in jumbled order. Have students rearrange the lines of code into the proper program (this is called a Parsons Problem).
- Flip your classroom: record your lectures, and have students watch them and take notes for homework. Any classwork drills or worksheets can be distributed for "homework," and the more complicated assignments that would normally be done at home, can be completed with your help when they come to class.

create printed-out sheets that students can write code onto. Class time should then be filled with reading assignments, and more complicated coding practice so you are available to tutor as needed.

- Encourage advanced students to take on additional programming challenges. One easy way to do this is to assign Programming Projects from the blue pages at the end of each Chapter. Unit 4 introduces the Magpie lab, a long form programming lab with plenty of enrichment opportunities; encourage students to work on this project if they are ever left with a few minutes after completing a class assignment.

1.6 Forum discussion

[Lesson 4.00 Test Review & Reteach \(TEALS Discourse account required\)](#)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 4.01 – Array Basics

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Define, populate, and access arrays.**

1.1.2 Assessments — *Students will...*

- **Complete** exercises with manipulatives on WS 4.1.

1.1.3 Homework — *Students will...*

- **Read** BJP 7.1 up to “For-Each Loop”
- **Complete** self-check questions \#1, 7, 9

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Classroom copies** of [WS 4.1](#), [Poster 4.2](#)
- **Array whiteboards** (see notes at end of lesson) **and dry-erase markers**
- **Student small-group assignments** (~3-4 students per group)
- **Large manipulative** for teacher demo (**optional**)

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction to arrays	20min
Student array activity	25min
Paper selection & grade announcement	5min

1.4 Procedure

Divide your class into small groups today, and have the manipulatives (see below) out at each group station, desk, or workspace. Offer a just-in-time intro to arrays as outlined below, but don't belabor instruction; students may have better success physically working through the activity. Circulate around the room to check for understanding, but let students help and challenge each other.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction to Arrays [20 minutes]

1.4.2.1 Emphasize with students...

1.4.2.2 Content - Uses of pre-built data structures

An array is a data structure that is available for a programmer to use in Java. You will find that there are situations where arrays will work perfectly for your program and they will facilitate finding a solution. Other times an array might not be the best data structure to use and you might have to use something different.

As you complete this activity, think carefully about how arrays could be used in your future programs.

1. Open with the example of daily temperature on the slides.

- Ask the class to build the program, they should reach the point where they calculate the average, but are unable to calculate the days above average as they must access the data a second time.
- In order to successfully make the program, we would need to store the temperature for every day.
- An array is an indexed structure that holds multiple values of the same type. Ask students if they've seen anything in Java that might be an array. (A String can be thought of as an array of characters!)
- The values stored in an array are called elements. Individual elements are accessed using an integer index (the position). Ask students what element is stored at index 2, 4, and 7 in this string/array.
- Since an array is an object, you must construct it (you can't just declare it as a variable).

```
int[] numbers = new int[10];
```

//

Nothing is in the highlighted brackets because you're describing what type is going to be contained in the array.

```
int[] numbers = new int[10];
```

Here's the name of your array—in this case we're making an array of numbers.

```
int[] numbers = new int[10];
```

*We use the *new* keyword since we're constructing an object, then we tell Java how many elements we want to store in our array. In this case we want to store 10 numbers in the array.*

- Check for student understanding by asking students to tell you how to construct an array that holds 9 integers.

```
int[] numbers = new int[9];
```

- Alternatively, you can initialize an array by writing out the full array.

```
int[] numbers = {0, 1, 2, 3, 4, 5};
```

2. Both sample arrays are only ½ done right now—they're arrays filled with 0s because Java auto-initializes arrays to a default value of 0 (for char, double, and int) or false (for boolean). So our *number* array looks something like this:

```
+-----+  
| array `numbers`: | 0 | 0 | 0 | 0 |  
+-----+  
| index | 0 | 1 | 2 | 3 |  
+-----+
```

- To fill in this array, we need to fill in the values for each location:

```
numbers[0] = 27;  
numbers[3] = -6;
```

- Now the array looks like this:

3.

```
+-----+  
| array `numbers`: | 27 | 0 | 0 | -6 |  
+-----+  
| index | 0 | 1 | 2 | 3 |  
+-----+
```

- Now that students can create an array of integers, ask the students how would they create an array of temperatures

4. and calculate the average temperature? What intermediate step would be needed before calculating the average temperature?

5. Briefly touch on other types of arrays and common errors.

- You can have arrays of almost anything: String, double, boolean, etc. Examples of an instantiated double and Boolean array are on the slides. Ask the class what they must change to create these arrays.
- Cover the common index-out-of-bounds exception. If the program calls illegal indexes or indexes outside 0 and the array's length-1, Java will throw you an exception. (It's always nice to go over reasons for exceptions so you don't need to correct them all later!)

6. If you have a really big array, you can use a Scanner to grab values from user input, or you can autofill them with a loop:

```
for (int i = 0; i < age.length; i++){  
    age[i] = input.nextInt();  
}
```

As you move across the array (in this case to fill each element with a user-inputted value), we call this "array traversal." You'll need to do this a lot in the future, so you should put a general formula in your Tricky Code Cheat Sheet:

```
for (int i = 0; i < age.length; i++){\n    // do something with age[i];\n}
```



1.4.3 Student Array Activity [25 minutes]

1. Distribute white boards and markers to group workstations before students get seated.
2. If you feel that your students need the additional structure, assign groups to work together on Problem 2.
3. Walk around the room, spot-checking for student understanding and answering any student questions.

1.4.4 Paper Selection & Grade Announcement [5 minutes]

At the end of class, consider choosing one group's whiteboard to evaluate as a demonstration for the class.

1.5 College Board Topic Questions

After this lesson, students will be able to answer questions from the College Board Unit 6 Topic Questions 6.1 Array Creation and Access

1.6 Accommodation and Differentiation

Rather than assigning groups randomly or by ability, use tiered grouping as a differentiation strategy. Concepts/skills will be covered at different levels of complexity in response to diagnosed needs of each learner. Your tiered group assignments will probably end up changing from one exercise to another, since students' needs and strengths vary with instructional objectives and task types.

You can reduce paper waste and increase student engagement by creating an inexpensive classroom set of "array whiteboards" following these instructions:

1. Purchase panel board from your local hardware store (<http://tinyurl.com/zgtlbhr>)
2. Have the assistant cut the board into long strips that you can use as 1 dimensional array.
 1. If you like to use individual whiteboards to check for student understanding, you can have whiteboards cut to individual student squares, then have students line up the little white boards into a one-dimensional array for this exercise.
 2. If you plan on using these as array boards, you can either subdivide the boards into element-blocks with black electrical tape, or you can have your students draw the blocks in with their dry erase markers. (The latter option encourages students to construct arrays of different sizes.)
3. Your total number of array white boards should be:
$$(\# \text{ small groups in your classroom}) * 2 + 1 \text{ instructor array board}$$
3. Use these array-whiteboards to demonstrate the relationship between 1 (and later 2) dimensional arrays during this unit.

In mathematics, a **manipulative** is an object which is designed so that a learner can perceive some mathematical concept by manipulating it, hence its name. The use of manipulatives provides a way for children to learn concepts in a developmentally appropriate, hands-on and experiential way.

[TEST: hello world]

1.7 Teacher Prior CS Knowledge

Arrays in Java (and other object oriented programming languages) are classes. This brings the whole object oriented paradigm into play. When declaring an array variable, the variable is now a reference to an array object. In order to create an object from a class the programmer uses the new keyword. There is a distinction between the array reference and the array object. This programming construct allows for multiple reference to point to the same object.

1.8 Common Mistakes

Arrays common mistakes: <http://interactivepython.org/runestone/static/javareview/arraybasics/amistakes.html>

1.9 Misconceptions

- Students understanding of the difference between the index i and the content of the i^{th} element stored in $a[i]$.
- Loop bounds:
 - 0-based index arrays,
 - where arrays end, versus array length, and
 - what are the range of indices to express a particular array range.

1.10 Video

- BJP 7-1, *Array Simulation*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c7-1
- CSE 142, *Arrays* (1:35–26:06)
<https://www.youtube.com/watch?v=6M3KEfFpYiM&start=95>
- CS Homework Bytes, *Arrays, with Ariel*
<https://www.youtube.com/watch?v=PFohS2HvCgs>

1.11 Forum discussion

Lesson 4.01 Array Basics (TEALS Discourse account required)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 4.02 — For-Each Loop & Arrays Class

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Populate and access** arrays using a for-each loop

1.1.2 Assessments — *Students will...*

- **Complete** manipulatives exercises on WS 4.2

1.1.3 Homework — *Students will...*

- **Read** BJP 7.2 up to “Reversing an Array”
- **Complete** self-check questions #12-14

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Classroom copies** of [WS 4.2](#)
- **Poster 4.2**
- **Array whiteboards** (see notes at end of LP 4.2) **and dry erase markers**
- **Student small-group assignments** (\approx 3-4 students per group)
- **Large manipulative** for teacher demo (**optional**)

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction to Arrays	10min
Student Array Activity	35min
Paper selection & grade announcement	5min

1.4 Procedure

As with yesterday's class, divide your class into small groups, and have the manipulatives out at each group station, desk, or workspace. Offer a just-in-time intro to for-each loops, but do not get bogged down in the introductory lesson. Many students will learn from physically practicing the for-each control structure using the array whiteboards. Circulate around the room to check for understanding, but let students help and challenge each other.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction to Arrays [10 minutes]

1. Briefly introduce the **for-each loop**, **array initialization**, and the **Arrays class** before dismissing the class to work on their group activity:

If you want to access each element in the array without changing the values (to summarize or count them), you can access them using a for-each loop:

```
for (<type> <name> : <array>){
    <statement>;
    <statement>;
    ...
}
```

2. Place an array on the board, demonstrating the quick way to declare an array, and then illustrate how a for-each loop could be used to access the array:

```
int[] fallTemperatures = {55, 50, 59, 69, 48, 30, 48};
```

- o This initializes an array called fallTemperatures with 7 integer values. Ask students when they might see these temperatures in their region, and how the values in the array would differ during another season or in a different location. It may make sense in your region to change the array name to winterTemperatures or nightTimeTemperatures, etc.

```
for (int i = 0; i < fallTemperatures.length; i++) {
    if (fallTemperatures[i] > 32) {
        above++;
    }
}
```

- o This is our traditional loop, which traverses the array and sums up all the temperatures that are above freezing (we assume there is a method called "above" that keeps a running count of how many days were above 32°). We can express this same process with a for-each loop:

```
for (int i : fallTemperatures) {
    if (i > 32) {
        above++;
    }
}
```

```
        }  
    }  
  
    with the general form of:  
  
for (<type> <name> : <array>) {  
    <statement>;  
    <statement>;  
    ...  
}
```

- Make a point of having your students write a note to remind themselves that for-each loops cannot modify values within an array, only examine each value in sequence.

1.4.3 Student Array Activity [35 minutes]

1. Distribute white boards and markers or paper printouts to group workstations before students get seated.
2. If you feel that your students need the additional structure, assign groups to work together.
3. If you think students will need the extra guidance, help students together in a whole-group setting. To help students without giving them the answer outright, point out that:
 - a. `numbers[7]` evaluates to 0.
 - b. `numbers[numbers[7]]` → `numbers[0]` does NOT evaluate to 0.
 - c. Instead, the memory location `numbers[0]` (the index 0 of the `numbers` array) receives a value.
4. Walk around the room, spot-checking for student understanding and answering any student questions.

1.4.4 Paper Selection & Grade Announcement [5 minutes]

At the end of class, consider choosing one group's whiteboard to evaluate as a demonstration for the class.

1.5 College Board Topic Questions

After this lesson, students will be able to answer questions from the College Board Unit 6 Topic Questions 6.3 Enhanced for Loop for Arrays

1.6 Accommodation and Differentiation

If you have a student that would benefit from additional tactile or visuospatial learning exercises, invite him or her to create a classroom poster that contains the same information as Poster 4.2.

Rather than assigning groups randomly or by ability, use tiered grouping as a differentiation strategy. Concepts/skills will be covered at different levels of complexity in response to diagnosed needs of each learner. Your tiered group assignments will probably end up changing from one exercise to another, since students' needs and strengths vary with instructional objectives and task types.

1.7 Additional Resources

- CS Awesome Array Algorithms

<https://runestone.academy/runestone/books/published/csawesome/unit6-arrays/topic-6-4-array-algorithms.html>

1.8 Video

- CSE 142, *Array Traversal* (26:06–33:26)
<https://www.youtube.com/watch?v=6M3KEfFpYiM&start=1570>
- CSE 142, *For-Each Loop* (19:40–22:40)
<https://www.youtube.com/watch?v=6M3KEfFpYiM&start=1180>

1.9 Forum discussion

Lesson 4.02 For-Each Loop & Arrays Class (TEALS Discourse account required)

formatted by Markdeep_1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 4.03 – Printing, Searching, & Testing for Equality

1.1 Overview

1.1.1 Objectives – *Students will be able to...*

- **Manipulate** single-dimension arrays using a variety of array transversal algorithms.

1.1.2 Assessments – *Students will...*

- **Teach** a mini-lesson on printing, searching/replacing, testing for equality, reversing an array, or string traversal.
- **Complete** a quiz at the end of Day 2

1.1.3 Homework – *Students will...*

- **Day 1:** - Read BJP 7.2 up to “Reversing an Array” Complete self-check questions #15-17 and exercise 3
- **Day 2:** - Finish 7.2 and complete self-check questions #19-21

1.2 Materials & Prep

- **Group copies** of [WS 4.3](#)
- **Assignments** for 5 student groups
- **5 classroom copies** of the textbook (or have students bring their copies to class)
- **Copies of the grading rubric** (on the overhead or printed out; **optional**)

You will need to circle student assignments on point 2 of WS 4.3, so each group knows what topic they are expected to teach.

1.3 Pacing Guide: Day 1

Section	Total Time
Bell-work and attendance	5min
Introduction to assignment	10min
Student preparation of lesson & quiz questions	40min

1.4 Pacing Guide: Day 2

Section	Total Time
Group set-up, attendance	5min
Group presentations (~8 minutes per group)	40min
Quiz	10min

1.5 Procedure

Your hook for today's lesson is to turn the reigns over to students immediately. Have instructions printed out and sitting at teamwork stations (or on student desks). Encourage students to answer their own questions using the instruction sheet and textbook. Frequently asked questions and suggestions for student groups are included in Accommodations and Differentiation.

1.5.1 Bell-work and Attendance [5 minutes]

1.5.2 Introduction to Assignment [10 minutes]

1. Review the assignment as outlined on WS 4.3, and ask students to explain back to you what their group presentations need to include for full credit. If you need to, review procedures for group work, then split the class into groups.

If you feel it will benefit your class, reviewing or distributing the grading rubric before the assignment should be done at this point.

1.5.3 Student Preparation of Lesson and Quiz Questions [40 minutes]

Give students the class period to prepare their presentation and generate quiz questions for tomorrow's class.

- Use a timer and periodically announce how much time is left in class so students can pace themselves.
- Emphasize to students that they should primarily draw upon their textbooks to help them plan their lesson.
- Answers to commonly asked questions, and tips for the different groups may be found below:

1.5.3.1 FAQ: PRINTING AN ARRAY

1.5.3.1.1 Guidance for student teachers

1. If students are lacking direction, encourage students to refer to the textbook and trace the loop and predict the output of the code that prints the array “list” vertically.
2. A good “Tricky Code Cheat Sheet” tip can be found on the same page; any time you see a list of values separated by commas, students should remember to use a fencepost algorithm.

1.5.3.1.2 Common Questions/Answers

1. When would we encounter an empty array?

If you set array values to null, they’re actually empty (not just equal to 0 or false). Another way you could get an empty array is if you initialized an array of size 0; then there would be no elements at all!

2. Why do we have to include extra code to account for an empty array?

We put in “contingency code” to deal with an empty array to be thorough. If we don’t handle special cases correctly, Java will throw an exception, so we try to cover our bases and prevent that from happening.

1.5.3.2 FAQ: SEARCHING AND REPLACING

1.5.3.2.1 Guidance for student teachers

1. If students are having trouble outlining their lesson, encourage them to teach counting, then locating values in a list. Once they’ve covered those methods, they can finish up by tracing the sample method replaceAll and explaining the output.
2. A good “Tricky Code Cheat Sheet” tip would be the convention of returning –1 if a value is not found in the list.

1.5.3.2.2 Common Questions/Answers

1. Why do we use a for-each loop for counting occurrences, but not for finding values in a list?

*Neither one requires you to change the contents of the array! **If we’re finding an item in a list and returning its index, then we need to track indexes as we search. That’s done naturally with a regular for loop; with a for-each loop, we’d have to manually simulate a for loop to compute the index.*

2. Why is the method “count” initialized to zero?

This is setting up the loop to work correctly—review “priming the pump” analogy if needed.

1.5.3.3 FAQ: TESTING FOR EQUALITY

1.5.3.3.1 Guidance for student teachers

1. Remind students to include the definition for what makes arrays equivalent (they have the same length and store the same sequence of values).
2. If students are struggling with the sequence of their lesson, you might suggest that they begin by teaching the sample method “equals,” then circling back to explain why testing for inequality is easier than testing for equality.
3. A good general tip for the Tricky Code Cheat Sheet would be the common pattern for methods like “equals;” Test all for the ways that the two objects might not be equal, returning false if there are any differences, or true at the very end if all tests are passed.

1.5.3.3.2 Common Questions/Answers

1. Can't we just use Arrays.equals to see if two arrays are equal?

Yes, you can (and should) use this method from the Arrays class. In teaching your unit, your purpose is to familiarize students with the way to write code that tests equals in general. While you might not need it in this particular situation, you may want to tweak the method down the road, and you cannot edit the Arrays.equals method because its not in a class you can edit.

1.5.3.4 FAQ: REVERSING AN ARRAY

1.5.3.4.1 Guidance for student teachers

1. If students need guidance on structuring their lesson, encourage them to work through the final (correct) method “swap,” trace the flow of control and output, then cover why other versions don’t work. (They may not present in this order, but covering the correct answer should help them organize their thoughts.)
2. A good tip for the Tricky Code Cheat Sheet is to summarize the steps needed to swap two values. This comes up often on the AP exam.

1.5.3.4.2 Common Questions/Answers

1. Why can't we just swap values by assigning them to each other?

If you assign one value to the other, you copy over the values, so ½ way through the process, you’ve got 2 copies of the same value. The second half of the swap doesn’t work!

2. Why do we have to stop the swap loop ½ way through the list.length? Won’t that just swap ½ of the list?

Using a line of 4 – 6 objects (pens, paperclips, whatever you have around the classroom), start by swapping the first & last items, then the next-inner-two, and so on. Count each move as you make it, then pause at the halfway point, pointing out you’ve conducted $\text{length}/2$ swaps. Proceed with the final swaps, returning objects back to their original place.

1.5.3.5 FAQ: STRING TRAVERSAL ALGORITHMS

1.5.3.5.1 Guidance for student teachers

1. Encourage the group to spend some time during their lesson on why you use parentheses for most arrays, but not with strings.

2. If students are at a loss for how to teach their segment, suggest covering an example and a non-example for contrast. A good non-example would be traversing an array that isn't a string.

1.5.3.5.2 Common Questions/Answers

None.

1. Collect quiz questions before the end of class. Check and compile into a quiz for the end of Day 2.
2. On Day 2, give each group 5 minutes to present their topic and 3 minutes for questions.
Encourage students to ask questions, and be sure to ask a question or two of each team (depending on how many teams you have).
3. Use the grading rubric as outlined here:

3 pts.	2 pts.	1 pts.	0 pts.
Presentation includes definitions and an example with proper syntax.	Presentation includes definitions or an example with proper syntax.	Presentation includes definitions or an example with proper syntax with few mistakes.	Presentation includes definitions or an example with proper syntax with many mistakes.
Presentation includes a non-example as helpful contrast.	Presentation includes a non-example that is marginally helpful.	Presentation includes a non-example that does not add to comprehension.	Presentation includes a non-example that adds confusion, or presentation does not include a non-example.
Presentation includes a helpful tip that is clearly explained and concisely stated.	Presentation includes a helpful tip that is clearly explained or concisely stated.	Presentation includes a helpful tip that is not clearly explained and may include a small error.	Presentation does not include a helpful tip or hint.

4. Administer the student-generated quiz to assess student understanding.

1.6 College Board Topic Questions

After this lesson, students will be able to answer questions from the College Board Unit 6 Topic Questions 6.2 Traversing Arrays

1.7 Accommodation and Differentiation

Circle around the room to help students through reading the text in the textbook. Make sure that each of your working teams are properly stratified (rather than using tiered grouping).

If students are speeding along, encourage students to write down questions to pose to other groups during mini-lessons. If everyone finishes creating their lessons early, start the classroom presentations on Day 1 instead of waiting for Day 2. If only 1 or 2 groups have finished early, encourage groups to rehearse lesson delivery.

1.8 Video

- BJP 7-2, *Array Traversal Algorithms*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c7-2
- CSE 142, *Random Access to Array* (33:26-42:00)
<https://www.youtube.com/watch?v=6M3KEfFpYiM&start=2006>

1.9 Forum discussion

Lesson 4.03 Printing, Searching, & Testing for Equality (TEALS Discourse account required)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 4.04 – Reference Semantics

1.1 Overview

1.1.1 Objectives – *Students will be able to...*

- **Compare and contrast** how primitives and arrays are treated when passed as parameters.

1.1.2 Assessments – *Students will...*

- **Complete** graphic organizers and a worksheet
- **Extra credit:** complete a Pokémon Challenge

1.1.3 Homework – *Students will...*

- **Read** BJP 7.3 and 7.4 up to “Command-Line Arguments”
- **Complete** exercises \#9, 10

1.2 Materials & Prep

- **Projector and computer** with this page: <http://www.legendarypokemon.net/javacalc.html>
- **Whiteboard and markers**
- **Classroom copies** of [WS 4.4](#)
- **Instructor copy** of WS 4.4 Answer

The “worksheet” for today is a 5-page work packet, so if your school has long lines/production time for the copy machine, plan ahead!

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Student activity	25–30min
Student trade & check	5min
Whole group review & paper submission	15min

1.4 Procedure

Hook your class today with the concept of a “backwards” class structure. Using only the information gleaned from last night’s reading (and perhaps some help from a friend), students should work through as much of the worksheet as they can in the time allotted (~30 minutes). Finish the class with whole-class note taking on the topics that were challenging in the sheet.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Student Activity [25-30 minutes]

Have students start working on WS 4.4 in pairs or alone. Use a timer to help students pace themselves.

1.4.3 Students Trade & Check [5 minutes]

After 25–30 minutes give students a few minutes to check each others’ work.

1.4.4 Whole Group Review & Paper Submission

1. As a whole group, ask students for questions they had on the worksheet. Use the answer key included on this to guide instruction.
2. Collect worksheets at the end of class.

1.5 Accommodation and Differentiation

If you have been using Parson Problems throughout the year, your students will be familiar with the format of Question 8 on the worksheet. For other classes, this may be the first time they’ve been asked to rearrange provided code. Read through the problem out loud with the class, then read through the lines of code in the bottom half of the question. Each line of code (even the lone bracket) can be shuffled and re-arranged to provide the correct code sequence.

Student success in this lesson relies heavily on students’ having been able to read and comprehend the prior nights’ reading. In ELL classrooms, encourage students to open their books and work with the text in front of them, and pair students of differing language abilities.

- If you know your students’ reading abilities will not allow for a lesson like this, conduct the lesson as a whole-group, teaching a segment of the chapter and pausing to let students work on a question before moving forward.

- The worksheet matches up sequentially with section 7.3, so you can have students read along with your in the book as you work through the sheet, and/or you can allow advanced students to work on their own as you help the rest of class.

There will probably be a lot of variation in how long it takes students to complete today's assignment.

- Ask students who finish early to design a hands-on demonstration that uses the array whiteboards (and any other materials around the room) to explain the proper answers to the questions on the worksheet. If they come up with any cool demos, use them during student review at the end of the class.
- Be ready with a Pokemon Challenge for the students that speed through the assignment:

1.5.1 POKEMON CHALLENGE

This java based calculator (<http://www.legendarypokemon.net/javacalc.html>) uses median IVs (initial values) and input EVs (effort values) to calculate a Pokemon's stats on a given level.

Write a method called medianIV that accepts an array of integer IVs as its parameter and returns the median of the numbers in the array.

The median number is the number that appears in the middle of the list if you arrange the elements in order. You can assume that the array is of odd size (so that one element is found in the middle), and that the numbers in the array are between 0 and 99, inclusive. For example, the median of [5, 9, 4, 10, 11] is 9, and the median of [0, 8, 1, 89, 48, 27, 30] is 27.

Hint: Check out the Tally program in chapter 7 for some ideas on what code to use.

1.5.1.1 Final Project

The Java based calculator that uses median IVs and input EVs to calculate a Pokemon's stats on a given level provides you with an in depth look of how complex some calculations can get in these types of games. As you continue to think about your final project in the course, consider how you will use calculations to determine a characters health or attack values.

1.6

1.7 Video

- CSE 142, *Array Initializer* (3:26–5:43)
<https://www.youtube.com/watch?v=DieHL2VO83k&start=206>
- CSE 142, *Passing array as parameter* (5:44–9:05)
<https://www.youtube.com/watch?v=DieHL2VO83k&start=344>
- CSE 142, *Arrays.toString()* (21:19–25:27)
<https://www.youtube.com/watch?v=DieHL2VO83k&start=1278>
- CSE 142, *Values vs reference* (25:28–39:18)
<https://www.youtube.com/watch?v=DieHL2VO83k&start=1528>
- CSE 142, *modifying array when passed as parameter* (39:19–43:41)
<https://www.youtube.com/watch?v=DieHL2VO83k&start=2359>

1.8 Forum discussion

formatted by Markdeep_1.093 ↗

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 4.05 – Shifting Values & Arrays of Objects

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Shift** elements within an array
- **Construct** arrays of objects

1.1.2 Assessments — *Students will...*

- **Complete** Practice questions
- **Model** memory manipulation using array whiteboards

1.1.3 Homework — *Students will...*

- **Read** BJP 7.4 “Nested Arrays” and BJP 7.5 “Rectangular Two-Dimensional Arrays”
- **Complete** self-check questions #27-29 and exercise #4

1.2 Materials & Prep

- Day 1
- **Projector and computer** with this page: <http://www.legendarypokemon.net/javacalc.html>
- **Classroom whiteboard & markers**
- **Array whiteboard & markers**
- **Small group assignments & seating arrangements** (if possible)
- Day 2
- Teacher access to CS Awesome **[Lesson 6.4: Array Algorithms]** Sign up at [CS Awesome AP CSA Java Curriculum]
- Access to Dr. Nguyen **[For Loops]** slide deck
- Access to CS Awesome **[6.4. Array Algorithms]**

If your classroom structure allows it, arrange array whiteboards and markers at stations where students can gather around them to model the concepts you teach during your intro. Ideally, students will be standing huddled around the arrays, actively moving and rearranging items.

1.3 Pacing Guide: Day 1

Section	Total Time
Bell-work and attendance	5min
Introduction & small-group practice	20–25min
Practice Questions	25–30min
Error-checking algorithm (see below)	10min

1.4 Pacing Guide: Day 2

Section	Total Time
Bell-work and attendance	5min
CS Awesome Array Algorithms 6.4	35–45min

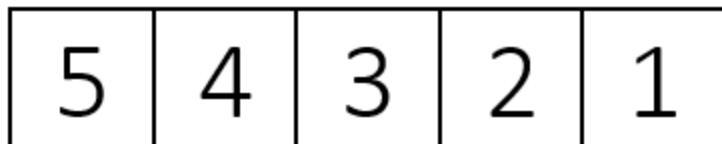
1.5 Procedure 1

Hook your class today by having them in small groups with the array whiteboards during your Introduction. As you write out the code samples on the whiteboard, pause at different stages to model what is happening to the elements stored in memory. As the introduction progresses, ask students to model the changes for the whole class, with you and the other groups replicating their manipulations. The key to this introduction is to have students working through the examples in physical space, so make sure that all groups are working along with their array whiteboards.

1.5.1 Bell-work and Attendance [5 minutes]

1.5.2 Introduction & Small-Group Practice [10 minutes]

1. Introduce your example array called metroCardRides (or dailyDrive, milesWalked, whatever represents your students' commute to school), and hold up an array of ints:



2. Briefly ask your students to direct you in constructing and initializing this array object, then ask students for some ideas as to how to move the 5 from the first element of the array to the last element, so that we end up with [4][3][2][1][5].

Give students a minute to try moving numbers around their whiteboards. Most numbers can be moved by shifting down the line with a for loop, but the first element will need to be removed to allow "space" for the shift.

3. Ask students how to store that first element (value 5) into a local variable, then write that code on your main whiteboard.

```
int first = metroCardRides[0];
```

Redirect students to the task of shifting the rest of your values on your array. As a class, discuss what you need the code to do, then ask the groups to draft some code and test that code on their array by tracing the code and checking for errors.

4. The correct loop for this loop looks like this:

```
for (int j = 0; j < metroCardRides.length - 1; j++) {  
    metroCardRides[j] = metroCardRides[j + 1];  
}
```

If students use metroCardRides.length instead of metroCardRides.length - 1, walk through the loop on your array whiteboard, demonstrating that metroCardRides.length causes Java to run off the end of the loop since there is no element at index 5. Alternatively, remind students about zero indexing, and see if they catch their own mistake.

5. Ask students to remind you (in pseudocode) what all the steps are that are needed to successfully shift the numbers in the array.

- Model on your array whiteboard what steps have already been successfully written, and ask students what is left to do now that your array looks like this:



- Select a group to give you the last step that re-inserts the first element at index 4, and ask the class to finish up the complete method, so your final method looks like this:

```
public static void firstToLast (int[] metroCardRides) {  
    int first = metroCardRides[0];  
    for (int j = 0; j < metroCardRides.length - 1; j++) {  
        metroCardRides[j] = metroCardRides[j + 1];  
    }  
    metroCardRides[metroCardRides.length - 1] = first;  
}
```

6. If all students understand this method, move the last element to the first position, shifting the other elements one position to the right.

- Direct students to start working on this challenge by re-setting their whiteboards and modeling the different steps that need to occur.
- Walk around the room, checking that students are on the right path. Students should recognize that they need to temporarily store the last element as a variable now that they're shifting array elements in the other direction.

7. Once students have written the code they think is correct, have them trace their own code and manipulate the arrays in their whiteboard.

Encourage students to divide up this task so one student reads the code and another student moves the elements on the whiteboard. All members should be engaged in error checking.

8. The correct final code looks like this:

```
public static void firstToLastRight (int[] metroCardRides) {  
    int last = metroCardRides.metroCardRides.length - 1;  
    for (int j = metroCardRides.length - 1; j >= 1; j--) {  
        metroCardRides[j] = metroCardRides[j - 1];  
    }  
    metroCardRides[0] = last;  
}
```

9. Spot check student code for an off-by-one error; starting the loop at 0 asks Java to look for a value at index -1 , which doesn't exist. (If students are modeling their code execution with the array whiteboards, they should catch this.)

- o If students “correct” this error by starting the loop at 1, ask them to model the code execution with their array white boards for you. The loop overwrites the value at $j - 1$ with the value at j , so the array will start filling in with all 5s as the loop repeats itself:

5	5	5	5	5
---	---	---	---	---

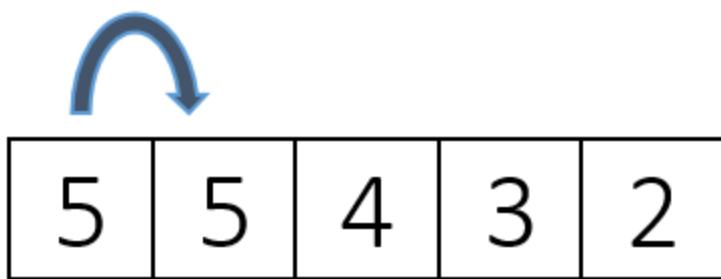
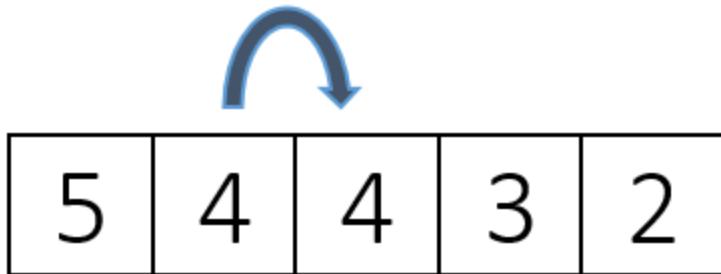
- o The solution to this is tricky! It might be worth discussing/working as a class if groups are getting stuck on this. The loop needs to shift values right by starting at the left and running backwards. See if you can get students to brainstorm this solution by manipulating the values on their whiteboards first.



5	4	3	2	2
---	---	---	---	---



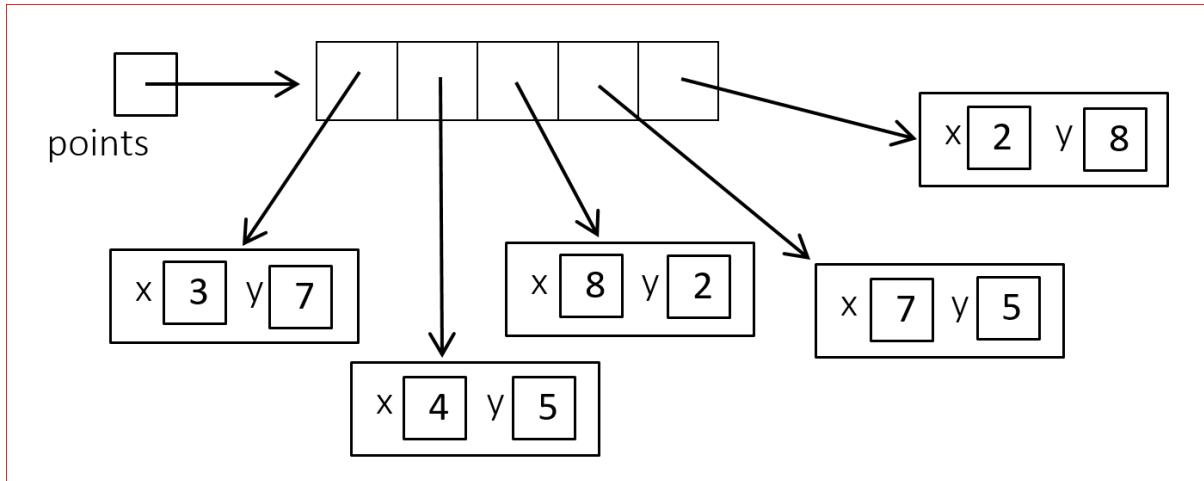
5	4	3	3	2
---	---	---	---	---



- Now add back in the temporarily-stored value 1 at index 0.

10. Collect the array whiteboards (or have students set them aside) and briefly introduce arrays of objects.

- This is an optional extension.** Since we haven't studied objects yet, you might want to leave this out if it is too far beyond your students' zone of proximate knowledge.
- Arrays of objects store reference to objects instead of a primitive type value. Drawing something like this might help student comprehension:



- Ask students leading questions, getting them to tell you that because the array and the array contents are both objects, both need to be constructed.
- Work through an example together as a class; ask students to help you construct an array of points. Your code should look something like this:

```
Point[] points = {
    new Point(3, 7),
    new Point(4, 5),
    new Point(8, 2),
    new Point(7, 5),
    new Point(2, 8)
};
```



1.5.3 Practice Questions [25-30 minutes]

1. Depending on the mood and frustration levels in the class, you may choose to have students work in pairs.
 - a. If students are really having a rough time, work through the first Practice question together as a whole group.
 - b. Put soft, soothing (but upbeat) music on in the background to encourage work
2. Have students complete the following Self Check practice problems at the back of the chapter.
 - a. Self-Check 7.22: arrayCodeTracing3 b. Self-Check 7.23: arrayCodeTracing4 c. Self-Check 7.25: arrayMystery2 d. Self-Check 7.26: arrayMystery3 e. Self-Check 7.30: isPalindrome
3. If more than 25% or more of the class is struggling, return to whole group with the stipulation that students who understand the task may continue working independently. Otherwise, encourage peer tutoring, or using the textbook or notebooks for help.

1.5.4 Error-Checking Algorithm [10 minutes]

At the end of class, if time permits, have students draft an algorithm for error-checking (see below for details).

1.6 Accommodation and Differentiation

If your class needs additional challenges, make the arrays in your examples a bit more sophisticated. Change the values or add “tricks” to the code, and include some mistakes in your delivery (written or conceptual) to let the students “catch” you. If they don’t catch you (and you’re a good actor), fumble your way into revealing the mistake in logic or code, and let them catch you so you can model a positive attitude towards error checking.

If students finish this lesson early, have them **draft an algorithm or checklist** that will help students check their loop code when shifting values in an array (what are common errors to look for?).

If the checklist is thorough, brief, and complete, have students create a handout or poster that you can hang up or share with the class.

To adjust for reading/comprehension challenges in the ELL classroom, consider having students work in small groups with the array whiteboards. Give each group/student a print out with the questions, and have them turn in the worksheets at the end of class.

1.7 Procedure Day 2

This lesson continues with array algorithms with examples from the College Board AP CS A exams free response questions using CS Awesome's Lesson 6.4 Array Algorithms.

1.7.1 Bell-work and Attendance \[5 minutes\]

1.7.2 CS Awesome Array Algorithms 6.4 \[35-40 minutes\]

The student lesson for Part 2 uses CS Awesomes [Lesson 6.4: Array Algorithms]. There you will find the lesson plan and activities to check for student understanding. The teacher lesson plans are accessible by joining the [Teaching CSAwesome google group] and navigating to [6.4. Array Algorithms]. The slide deck for this lesson are located on [Dr. Long Nguyen] GitHub at [For Loops]

1.8 College Board Topic Questions

After this lesson, students will be able to answer questions from the College Board Unit 6 Topic Questions 6.4 Developing Algorithms for Arrays.

1.9 Video

- BJP 7-3-1: *Array shifting algorithm*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c7-3-1
- CSE 142: *Mystery code walk through* (9:06–21:18)
<https://www.youtube.com/watch?v=DieHL2VO83k&start=560>
- CSE 142: *Reversing an array* (43:42–49:30)
<https://www.youtube.com/watch?v=DieHL2VO83k&start=2622>
- CSE 142: *Absolute values an array* (1:42–5:11)
<https://www.youtube.com/watch?v=DieHL2VO83k&start=102>

1.10 Forum discussion

Lesson 4.05 Shifting Values & Arrays of Objects (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 4.06 — Nested Loop Algorithms & Rectangular Arrays

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Correctly adjust** nested loop headers for use with arrays.
- **Correctly construct** two-dimensional arrays.

1.1.2 Assessments — *Students will...*

- **Complete** WS 4.6

1.1.3 Homework — *Students will...*

- **Read** BJP 10.1 up to “Adding to and Removing from an ArrayList”
- **Complete** self-check problems #1 – 6

1.2 Materials & Prep

Day 1 - **Projector and computer** - **Whiteboard and markers** - **Classroom copies** of [WS 4.6 - Array whiteboards](#)

Day 2 - Teacher access to CS Awesome [[Unit 8 Lesson 2 Traversing 2D Arrays \(nested loops\)](#)] Sign up at [CS Awesome AP CSA Java Curriculum]

- Access to CS Awesome [[8.2.4. Enhanced For-Each Loop for 2D](#)], [[8.2.5. 2D Array Algorithms](#)]

1.3 Pacing Guide Day 1

Section	Total Time
Bell-work and attendance	5min
Introduction/Review of objects & string processing	≈10min
Round Robin	≈35min
Paper selection & grade announcement	3min

1.4 Pacing Guide Day 2

Section	Total Time
Bell-work and attendance	5min
CS Awesome Enhanced for and algorithms	35-40min
Wrap-up	5min

If the instruction takes longer than expected, assign the worksheet to small groups so students can work together collaboratively. This should allow you to complete the lesson in one class period. Alternatively, you can assign students to finish the worksheet at home for homework, or make the worksheet shorter.

1.5 Procedure Day 1

There are several ways you can teach today's class. You should first check in with your students to see how prepared they are for today's lesson. If students understood most of what they read for homework last night, you can ask students for specific questions, cover only those topics, then move on to the Round-Robin activity. If your class is mostly confused, you can re-teach all of the content, working through several examples with Think Pair Shares before breaking into the Round Robin Activity. An optional Programming Challenge is included at the end of the lesson if your lesson finishes early.

1.5.1 Bell-work and Attendance [5 minutes]

1.5.2 Introduction/Review of Objects & String Processing [10 minutes]

1.5.2.1 Emphasize with students...

1.5.2.2 Content - Advanced programming structures - Uses of pre-built data structures

Multidimensional arrays have quite a few uses. They can serve as grids for seating plans or for board games such as chess and checkers. Some students enjoy recreating the game of battleship using two dimensional arrays.

As you learn about their use, remember to consider how you might use multidimensional arrays in your future projects.

1. Two dimensional arrays come in handy when tracking certain types of data; adjust the explanation so it's relevant to your students (a grocery store, a Starbucks, a farmer's market).
 - o Previously, we might have used an array to keep track of sandwich orders in line at a bodega:

turkey	ham	roast beef	turkey
--------	-----	------------	--------

- o Suppose we have several workers at the bodega making sandwiches, and we want to keep track of the orders each one is working on. We could use multiple arrays, or an array of arrays.

turkey	ham	roast beef	turkey
ham	turkey	roast beef	chicken
roast beef	roast beef	turkey	roast beef

- o To traverse multidimensional arrays, we need a new tool to help us fill them.

2. Demonstrate a basic nested loop used to switch numbers in an array. Begin by using the array whiteboard to demonstrate what you want the code to do:

4	3	2	1
---	---	---	---

- o Challenge the students to come up with pseudocode (or actual code, if they're ready for it) that prints out all of the inversions in this array. An inversion is a pair of numbers in which the first number in the list is greater than the second number. Be sure to emphasize that no elements are being moved or shifted. The output for the array above would be:

(4, 3)
(4, 2)
(4, 1)
(3, 2)
(3, 1)
(2, 1)

//

- o An array

3	1	4	2
---	---	---	---

would output:

(3, 1)
(3, 2)
(4, 2)

//

3. As your students are working on this problem, encourage them to use the whiteboard arrays to organize their thoughts and visualize the problem. Their pseudocode should look something like this:

```
for (every possible first value) {  
    for (every possible second value) {  
        if (first value > second value) {  
            print (first, second).  
        }  
    }  
}
```

//

- o As a whole group, or in individual groups (depending on familiarity with the material), construct the final code:

```
for (int i = 0; i < data.length - 1; i++) {  
    for (int j = i + 1; j < data.length; j++) {  
        if (data[i] > data[j]) {
```

```
        System.out.println ("(" + data[i] + ", " + data[j] + ")");
    }
}
```

//

4. As a “Tricky Code Cheat Sheet” tip, you might discuss the general reminder that for an inversion, the second value has to appear after the first value in the list. This means that you only want to compare values that come after the first value, so the inner loop initializes at $j = i + 1$.

If students ask why the outer loop ends at `data.length - 1`, have them trace the code and predict output using their whiteboards. They should notice that since only pairs of numbers are being compared, the last element of the array will never be a possible first value.

5. Ask a volunteer student to help you build a physical representation of a multidimensional array (an array of arrays). Demonstrate the code used to create arrays of multiple dimensions, and have students gather array whiteboards from around the room to show what the arrays would look like in memory:

- double: one double value

1.5

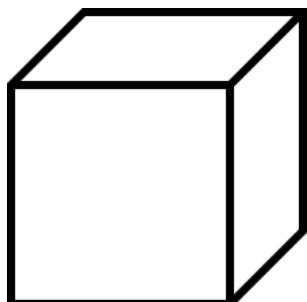
- double[]: a 1 dimensional array of doubles

3	1	4	2
---	---	---	---

- double[][]: a 2 dimensional array (grid) of doubles

3.5	7.1	0.4	92.8
6	8.7	3.3	-0.2

- double[][][]: a 3 dimensional array (cube) of doubles



- An array constructed with the code below has 2 rows and 3 columns:

```
double[][] ages = new double[2][3];
```

//

Have students index the array for you:

	0	1	2
0	0.0	0.0	0.0
1	0.0	0.0	0.0

- You might opt to give students this general formula for the syntax of declaring and constructing a multidimensional array:

turkey	ham	roast beef	turkey
--------	-----	------------	--------

6. Run through some examples with your students:

- To access the 1st element of the 2nd row: ages[1][0]
 - To access all elements of the 1st row: ages[0]
 - To access all elements of the 2nd row: ages[1]
7. Multidimensional arrays can also be passed as parameters. Have students trace the flow of control and predict the output of the code below:

```
public static void print (double[][] grid) {  
    for (int i = 0; i < grid.length; i++) {  
        for (int j = 0; j < grid[i].length; j++) {  
            System.out.print(grid[i][j] + " ");  
        }  
        System.out.println();  
    }  
}
```



- Ask students why you referenced grid.length in the outer loop (the number of rows), and what grid[i].length refers to (the number of columns).
- As a final tip, let students know that for multidimensional arrays, Arrays.toString won't work correctly, and they should use Arrays.deepToString instead.

1.5.3 Round Robin [45 minutes]

1. Round-robin is a drilling and error-checking exercise used with worksheets. At minimum, there should be 1 question for each student (e.g. a class of 15 students would need a worksheet with 15 or more questions). Students write their name on the worksheet, complete the first problem, then pass the paper to the student on the right (or whatever direction you choose). The next student first checks the previous answer, correcting it if need be, then completes the second question. Each student then passes on the paper again. By the end of the exercise, each student has checked and completed each question on the worksheet.
2. The hook is that you choose only ONE worksheet from the pile to grade. All students get a grade from that one worksheet. This keeps students invested throughout the exercise. Advanced students will check questions throughout the whole worksheet, and all students will try their best to catch their own (and others') mistakes, since the whole class shares the randomly-selected-paper's grade.
3. You should time each question/checking interval, and call "TIME!" when it is time for students to pass along papers.
 - a. Questions 1–4 should take ≈1 minute each.
 - b. Questions 5–9 should take ≈2 minutes each.
 - c. Questions 10–12 should take ≈3 minutes each.
 - d. Questions 13–14 should take ≈4 minutes each.
 - e. Questions 15–16 [Bonus] should take ≈5 minutes each.

Adjust the timing on these questions as needed, but try to keep a brisk pace. Part of the engagement factor is the sense of urgency.

1.5.4 Paper Selection & Grade Announcement [3 minutes]

If time allows, randomly select the worksheet and announce the class grade with a bit of fanfare, congratulating the class on a job well done. If there are any incorrect answers, use the time at the end of class to review the correct solutions or take questions.

1.6 College Board Topic Questions

1.7 Accommodation and Differentiation

To optimize this exercise, you might consider rearranging students (or creating a passing-path) that mixes students of different coding abilities. The advanced students can use the extra time to correct mistakes made by others; if they are sitting in proximity to the student that made the error, they will have a better chance of explaining the correct answer to them.

Due to the brisk pace of the round-robin rotation, there shouldn't be too much down time for any one student. If your students are finishing faster than the time intervals indicated, reduce the amount of time allotted to maintain a sense of urgency/keep students on task. If finished early, offer a Programming Challenge while you grade the sample worksheet.

1.8 Procedure Day 2

1.8.1 Bell-work and Attendance \[5 minutes\]

1.8.2 CS Awesome Enhanced for and Array Algorithms 6.4 \[30-40 minutes\]

The student lesson for Part 2 uses CS Awesomes [8.2.4. Enhanced For-Each Loop for 2D], [8.2.5. 2D Array Algorithms]. There you will find the lesson plan and activities to check for student understanding. The teacher lesson plans are accessible by joining the [Teaching CSAwesome google group] and navigating to [Unit 8 Lesson 2 Traversing 2D Arrays (nested loops)].

Students will log into [8.2.4. Enhanced For-Each Loop for 2D] and complete activities 1-4.

1.9 Common Mistakes

Two dimensional arrays common mistakes:
<http://interactivepython.org/runestone/static/javareview/array2dbasics/a2dmistakes.html>

1.10 Videos

- BJP 7-3-2, *Tallying with an Array*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c7-3-2

1.11 Forum discussion

Lesson 4.06 Nested Loop Algorithms & Rectangular Arrays (TEALS Discourse account required)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 4.07 — ArrayList

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Construct** code using ArrayList.
- **Predict** the output of methods that take arrays as parameters and/or return arrays.

1.1.2 Assessments — *Students will...*

- **Evaluate** statements and **predict** output during a game of grudgeball

1.1.3 Homework — *Students will...*

- **Read** BJP 10.1 up to “Adding to and Removing from an ArrayList”
- **Outline** Chapter 7 and BJP 10.1 “ArrayList”
- **Complete** self-check questions \#3-6 and exercise \#3

1.2 Materials & Prep

- **Projector and computer** (optional)
- Day 1
- **White paper and markers**
- **Classroom copies** of [Poster 4.7](#)
- **Rules** for grudgeball (see website for details: <http://toengagethemall.blogspot.com/2013/02/grudgeball-review-game-where-kids-attack.html>)
- **Team assignments** that divide your class into 5 or 6 teams
- **Nerf hoop & ball** (or wastepaper and trash can)
- **Taped 2- and 3-point lines**
- Day 2
- Teacher access to CS Awesome [**Unit 7 Lesson 4 ArrayList Algorithms Lesson Plan**] Sign up at [CS Awesome AP CSA Java Curriculum](#)
- Access to CS Awesome [7.4. ArrayList Algorithms](#)

Briefly review the rules of Grudgeball if you have forgotten them. If you have removed your 2 and 3 point lines from last time you played, test out your 2 and 3 point lines before class begins.

1.3 Pacing Guide: Day 1

Section	Total Time
Bell-work and attendance	5min
Introduction and note-taking	15min
Grudgeball	35min

1.4 Pacing Guide: Day 2

Section	Total Time
Bell-work and attendance	5min
CS Awesome Activities from Lesson 7.4	35–45min

1.5 Procedure: Day 1

To hook your class for today's material, and if space and whiteboard setup allow, set up the grudgeball "court" and scoreboard before class begins. Remind students that lecture content will be tested during the game.

1.5.1 Bell-work and Attendance [5 minutes]

1.5.2 Introduction and note-taking [10 minutes]

1. Ask students to name some limitations of arrays: shifting values is an ordeal, adding elements requires forming a new, larger array and copying values over, deleting elements leaves empty, unused indexes.
2. Introduce the more flexible `ArrayList` (be sure to remind students that they need to `import java.util.ArrayList`):
 - Uses arrays to store values (fast random access)
 - The `ArrayList` class contains methods to make add, remove, and shift values easily.
 - `ArrayList` takes a type parameter to determine what kind of values it will use as elements:
 - `ArrayList<string>` stores a list of Strings.
 - `ArrayList<point>` stores a list of Points.
 - If you forget to pass a parameter with the type you want the array to contain, the code won't execute.
3. Construct an `ArrayList` of Strings to demonstrate syntax:

```
ArrayList<string> spongebob = new ArrayList<string>();
```

- Even though the notation looks a bit different, the syntax is fairly similar to what we've used in the past. `ArrayList<type>` is how we indicate the type—just like you'd use `int` when declaring a one dimensional or two dimensional array.

```
ArrayList<string> spongebob = new ArrayList<string>();
```

- o This is the name of your ArrayList. It can be any non-keyword that you want to use.

```
ArrayList<string> spongebob = new ArrayList<string>();
```

- o Ask students if they can tell you what the new keyword is for (we use the new keyword when constructing an object).

```
ArrayList<string> spongebob = new ArrayList<string>();
```

- o Whenever you see empty parentheses, it means that you're not using parameters.

- o The ArrayList constructor ArrayList<>() constructs an empty list.

4. Using Poster 4.7, review some of the methods you can use to manipulate ArrayLists. Add some spongebob elements to your ArrayList:

```
spongebob.add ("Patrick Star");
spongebob.add ("Squidward Tentacles");
spongebob.add ("Mr. Krabs");
spongebob.add ("Pikachu");
spongebob.add ("Sandy Cheeks"); //
```

- o Ask students for suggestions on how to print out this ArrayList, and ask them to predict the output:

```
System.out.println("Some of the characters on Spongebob are " + spongebob); //
```

- o Students will probably notice that Pikachu is not a character in the Spongebob cartoon; ask them to refer to Poster 4.7 to suggest some code to remove Pikachu from the list:

```
spongebob.remove(3); // Pikachu is stored at index 3 //
```

- o Now ask students to add another character from the show to the middle of the list, at index 3:

```
spongebob.add(3, "Plankton"); //
```

- o The first parameter 3 indicates the target location, and the second parameter "Plankton" indicates the String to be stored there.

```
["Patrick Star", "Squidward Tentacles", "Mr. Krabs", "Sandy Cheeks"] //
```

becomes

```
["Patrick Star", "Squidward Tentacles", "Mr. Krabs", "Plankton"] //
```

5. Briefly review a few other useful ArrayList methods. Students will have an opportunity to practice (and you will have an opportunity to reteach if needed) during Grudgeball, so this can be a quick overview:

1.5.2.1 ARRAYLIST METHODS OVERVIEW

1.5.2.1.1 To get an element from the ArrayList and print it

```
System.out.println(spongebob.get(3)); //
```

1.5.2.1.2 Index Out of Bounds Exception

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: //
```

Since the indices for an ArrayList start at 0 and end at the number of elements - 1, accessing an index value outside of this range will result in an ArrayIndexOutOfBoundsException being thrown.

1.5.2.1.3 To get the number of elements in the ArrayList and print it

```
System.out.println(spongebob.size());
```

//

1.5.2.1.4 To add all the elements in the ArrayList

```
int sum = 0;  
  
for (int i = 0; i < spongebob.size(); i++) {  
    String s = spongebob.get(i);  
    sum += s.length();  
}  
System.out.println("Total of lengths = " + sum);
```

//

Have students justify your code choices, and ask a student (or students) to trace the code and narrate the steps for the class.

1.5.2.1.5 To replace an array element (no shifting)

```
spongebob.set(3, "Plankton");
```

//

This would replace Pikachu with Plankton directly, without requiring the shifting of the array.

1.5.2.1.6 To clear an array

```
spongebob.clear();
```

//

This removes all elements from the list and leaves null values at each index (it's an empty array now).

1.5.2.1.7 Enhanced for each loop

```
ArrayList<integer> intList = new ArrayList<integer>();  
intList.add(95);  
intList.add(85);  
int total = 0;  
for (Integer value: intList)  
{  
    total = total + value;  
}  
System.out.println(total);
```

//

Changing the size of an ArrayList while traversing it using an enhanced for loop can result in a ConcurrentModificationException being thrown. Therefore, when using an enhanced for loop to traverse an ArrayList, you should not add or remove elements.

1.5.3 Grudgeball [35 minutes]

1. Divide students into their assigned teams.
2. Review the rules for grudgeball, and have the students repeat the rules back to you.
3. Using the problems listed below (and any you may add, depending on your class' needs), play grudgeball until a team wins, or until the class period ends.
 - a. If a class gets the answer wrong, BRIEFLY pause the game to have students offer corrections before moving to the next team's question.

b. If correction seems to be dragging on, jump in and quickly re-teach using the incorrect answer as your example. It is important to keep the pace going to maintain student interest in the game!

Gudgeball problems & answers have been grouped assuming that you have 6 teams. If you have fewer teams, each “round” will be shifted accordingly, so you may have rounds where different teams are practicing different concepts. Judge each team’s knowledge gaps, and adjust which questions you ask each group accordingly.

1.5.3.1 GRUDGEBALL PROBLEMS AND ANSWERS

Use a type parameter to declare an ArrayList that:

- a. Stores a list of Strings → `ArrayList<string>`
- b. Stores a list of integers → `ArrayList<integer>` (Wrapper class)
- c. Stores a list of Points → `ArrayList<point>`
- d. Stores a list of doubles → `ArrayList<double>` (Wrapper class)
- e. Stores a list of soccer teams → `ArrayList<string>`
- f. Stores a list of temperatures → `ArrayList<double>` (Wrapper class)

Construct an ArrayList:

- g. Called z that stores a list of ints → `ArrayList<int> z = new ArrayList<integer>();`
 - h. Called list that stores a list of Strings → `ArrayList<string> list = new ArrayList<string>();`
 - i. Called jose that stores a list of Points → `ArrayList<point> jose = new ArrayList<point>();`
 - j. Called pokemon that stores a list of Pokémons → `ArrayList<string> pokemon = new ArrayList<string>();`
 - k. Called metroCard that stores the number of metrocard rides each student has left on their card today → `ArrayList<integer> metroCard = new ArrayList<integer>();`
-

1.6 College Board Topic Questions

After this lesson, students will be able to answer questions from the College Board Unit 7 Topic Questions 7.1 Introduction to ArrayList, 7.2 ArrayList Methods, and 7.3 Traversing ArrayList

1.7 Accommodation and Differentiation

In ELL classrooms, read the questions aloud in addition to showing the question on the board or projector. Consider distributing a worksheet with the questions on it so students can write down answers during the game.

1.8 Procedure Day 2

Students will be introduced to a variety of ArrayLists algorithms.

The student lesson for Part 2 uses CS Awesomes [7.4. ArrayList Algorithms]. There you will find the lesson plan and activities to check for student understanding. The teacher lesson plans are accessible by joining the [Teaching CSAwesome google group](#) and navigating to [Unit 7 Lesson 4 ArrayList Algorithms Lesson Plan](#).

1.9 Common Mistakes

ArrayList common mistakes: <http://interactivepython.org/runestone/static/javareview/listbasics/listmistakes.html>

1.10 Misconceptions

Java uses 3 different syntax for getting lengths which is a source of student confusion:

```
String.length()  
array.length  
ArrayList.size()
```



1.11 Videos

- BJP 10-1, *Removing from an ArrayList*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c10-1
- BJP 10-2, *Adding to an ArrayList of Integers*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c10-2
- CSE 142, *ArrayList* (6:40–25:29)
https://www.youtube.com/watch?v=-CX7aZGrc_k&start=400
- CSE 142, *Wrapper Class* (41:53–44:54)
https://www.youtube.com/watch?v=-CX7aZGrc_k&start=2510

1.12 Forum discussion

Lesson 4.07 ArrayList (TEALS Discourse account required)

formatted by [Markdeep 1.093](#) A small checkmark icon indicating the text has been formatted correctly.

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 4.10 — Review

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Identify** weaknesses in their Unit 4 knowledge.

1.1.2 Assessments — *Students will...*

- **Create** a personalized list of review topics to guide tonight's study session.

1.1.3 Homework — *Students will...*

- **Study** for tomorrow's test using targeted review list

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and marker**
- **Results** from electronic survey of review topics
- **Classroom copies** of the practice test [WS 4.10](#)

Once students have submitted their review requests, assemble those topics into categories and prepare to re-teach the topics as needed. If you are crunched for time, hold the review session during the second half of Day 5 of the Magpie Chatbot Lab.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Review of student questions	30min
Sample test review	15min
Check student study lists	5min

1.4 Procedure

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Review of Student Questions [30 minutes]

1. Clearly indicate that you expect all students to have a list of review topics to study this evening. Periodically remind students that this list will be checked at the end of class.
2. Begin with a review of student-submitted questions before reviewing the practice questions.

1.4.3 Sample Test Review [15 minutes]

1. Begin review with practice test WS 4.10. Have students work through section I questions, then review the answers as a class.
2. Give students time to complete section II questions, then review the answers as a class.
3. Finally, work through the various review topics, prioritizing questions that popped up the most.
 - o Some questions you may address while working through the sample test.
 - o Be ready for additional questions to pop up as you go. Save yourself the work and use old homework questions and student-generated test questions as examples to work through.
4. Use a combination of group-solving questions on the whiteboard, think-pair-share, and timed-response as review strategies.
5. After you've completed reviewing an idea, remind the class that they should write down that topic if they feel they still have to review it tonight.

1.4.4 Check Student Study Lists [5 minutes]

Spend the last 5 minutes of class checking each student's review topic list.

1.5 Forum discussion

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 4.XX — Programming Project (MagPie Alternative)

1.1 Overview

This project is intended to be an alternative to the MagPie Lab in the original AP computer science course. Starting from 2014-2015 the College Board announced that 20 hours of hands-on lab time may replace the original required labs (MagPie, PictureLab, Elevens). In this document, we recommend some open-ended alternatives.

1.1.1 Objectives — *Students will be able to...*

- **Conduct user-centered research** to identify specific functions for a specialized application
- **Plan and create** a calculator that perform specialized operations for an end-user
- **Test, evaluate, and share** the end product

1.1.2 Assessments — *Students will...*

- **Apply** if-else, String methods to implement a software application
- **Submit** a complete, functional program

1.1.3 Pacing Suggestion

- The duration of this project is at the discretion of the teacher. We recommend 2 hours for class time design and implementation, and 1-2 hours for out of class time to connect with end-user.
- Project involves conducting research work (survey or interviews), and communicating with end-user, out of the classroom.

1.2 Implementation Details

1.2.1 Complexity and Creativity

Students should come up with the idea themselves, based on user-centered research, and ideate a specialized application to address the needs of a specific user group.

Students will follow applied design process to implement the idea. You should talk to your teacher often to ensure that your progress is in-line with expectations.

1.2.2 Documentation and Style

As with all projects, your program must be well-written, well-documented, and readable. Writing code with good style is always a good idea. This will help you debug, pick up where you left off each day, and keep track of progress.

STEP 1 - UNDERSTANDING CONTEXT

Conduct user-centered research to find design opportunities and barriers.

Select an end-user for whom you will design and create this program (this can be a friend, classmate, relative, etc.). Create interview questions that will allow you to understand the end-user's interests and likes/dislikes.

Here are some possible applications that could potentially use "if" statements and "String" methods:

Word Games * Lipogram Word Game <https://en.wikipedia.org/wiki/Lipogram> * A Mad-Libs Program
https://en.wikipedia.org/wiki/Mad_Libs * A acrostic builder pyramid word builder
<http://www.crauswords.com/pyramidword.html>

Classic Ciphers with a Twist (eg, Morse Code, Caesar Cipher, etc.) https://en.wikipedia.org/wiki/Classical_cipher
* Create your own encryption program, but include some decision making (eg. Ask user for clues/hints for key generation?) * Create the corresponding decryption program, but include some decision making * Work in pairs! Test with one another.

Chatbot <https://en.wikipedia.org/wiki/Chatbot> * One of the four Activities in the original MagPie Lab (estimated 1-2 hour time) * A specialized, modern ChatBox (Siri/Alexa) for a specific end-user in mind:

- TV show related
- music DJ
- learning tool for ELL, French immersion, or other foreign language students
- someone who is color blind
- someone who owns a pet store

Other * Other application that involves STRINGS (creation, parsing, and processing), together with decision making (IF statements) - Medical terminology taxonomy - language learning applications - creating baby books or random poetry

After you have some initial ideas, interview some potential end-user to clarify your project specifications.

1.2.2.1 Emphasize with students...

1.2.2.2 Big Ideas - personal design interests require the evaluation and refinement of skills

The possibilities are endless. We live in a world that operates on the written or spoken word.

The computer and internet has forced us to digitize the words we use into zero's and one's for transmission purposes. But at a higher level, most people are working with "strings" (a collection of character symbols, making up a word). Sometimes words are easily understood and recognized, sometimes it is not.

Our human brains naturally make decisions and choices. With so many options, and paths, we can ask the computer to help us explore these choices. This is great when it comes to routine, repetitive, or time consuming tasks.

Poets, writers, travelers, wanderers, cryptographers, kings and queens have loved playing with words (strings, characters), and creating games (logic, secret messages) since the earliest of times to entertain, inspire, and protect the land.

Let your creative juices flow!

STEP 2 - DEFINING AND IDEATING

Choose a design opportunity and point of view, make inferences about limitations and boundaries. Take creative risks to identify gaps to explore, generate a range of possibilities, prioritize ideas for prototyping.

Using the responses from your end-user interview, begin to develop a plan for your custom application. Given the duration of the project, you should limit your calculator to do *an interesting activity* for a *specific end-user*. At this point you will only be able to implement a text-based user interface (i.e., no graphical elements, like buttons or scrollbars). Keep it simple and easy to use.

List:

- * the types of calculator functions it will do:
 - what will the user input be?
 - what output will be?
 - * what user interaction with the program will the user have?
 - how do you start the activity?
 - how does the activity end?

Share these ideas with your end user. Record their comments, suggestions and feedback and note any changes that you may make as a result of this interview.

Identify any issues or problems that might arise as you begin to program (what areas might require more information or programming solutions? Where can you find this information?)

CHECKPOINT 1

SUBMIT YOUR INTERVIEW QUESTIONS, RESPONSES AND PROGRAM DESIGN PLAN TO YOUR TEACHER.

STEP 3 – PROTOTYPING AND TESTING

Construct prototypes, making changes to code as needed.

Program your calculator. Be sure to review course notes and activities to make sure that you effectively implement object oriented programming design for your application.

When you are ready, create a short test plan and test your program. Include expected output and actual output and include details related to any fixes that needed to be made. A good motto to remember is **code a little, test a little**

Have your end-user try a working version of your program. Note their suggestions, comments and feedback. Indicate any changes that you made to the program, based on the user's feedback.

CHECKPOINT 2

SUBMIT THE CURRENT VERSION OF YOUR PROGAM TO YOUR TEACHER, AS WELL AS YOUR TEST PLAN AND FEEDBACK FROM THE END-USER

STEP 4 – SHARING, TESTING AND FINAL ITERATION

Gather feedback from users over time to critically evaluate your design and make changes to product design or processes Identify new design issues.

Share your work with other classmates, friends or family. Record any feedback, suggestions and comments and use this information to make the final iteration of your program.

CHECKPOINT 3

SUBMIT THE FINAL VERSION OF YOUR PROGAM TO YOUR TEACHER, AS WELL AS YOUR FEEDBACK FROM CLASSMATES, FRIENDS OR FAMILY.

Grading Scheme/Rubric

Implementation	
Project is appropriately complex and creative	4 points
Program is well-documented and shows good style	10 points
Final product meets all requirements and goals laid out in checkpoint specifications	8 points
Program uses programming concepts effectively, including all required elements with an appropriate level of complexity	4 points
Object Oriented Programming concepts are effectively applied with an appropriate level of complexity	6 points

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 5.04 — Encapsulation

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Explain** what encapsulation/abstraction are, and why they are important programming strategies.

1.1.2 Assessments — *Students will...*

- **Teach** a mini-lesson on encapsulation/abstraction, using private fields, using class invariants, and changing internal implementations

1.1.3 Homework — *Students will...*

- **Read** BJP 8.4
- **Complete** a quiz at the end of Day 2
- **Complete** chapter 8 self-check questions 17-21

1.2 Materials & Prep

- **Group copies** of [WS 5.4](#)
- **Assignments for 4 student groups**
- **4 classroom copies** of the textbook (or have students bring books to class)
- **Copies of the grading rubric** on the overhead or printed out (optional)

You will need to circle student assignments on point 2 of WS 5.4, so each group knows what topic they are expected to teach.

1.3 Pacing Guide: Day 1

Section	Total Time
Bell-work and attendance	5min
Introduction; reviewing the assignment	10min
Student preparation of lesson	30min
Student practice	10min

1.4 Pacing Guide: Day 2

Section	Total Time
Bell-work and attendance	5min
Student mini-lessons	35min
Quiz using student-generated questions	15min

1.5 Procedure

Your hook for today's lesson is to turn the reins over to students immediately. Have instructions printed out and sitting at teamwork stations (or on student desks). Encourage students to answer their own questions using the instruction sheet and textbook. Frequently asked questions and suggestions for student groups are included in Accommodations and Differentiation.

1.5.1 Bell-work and Attendance [5 minutes]

1.5.2 Introduction & Reviewing the Assignment [10 minutes]

1. Prepare students by telling them that the 4 topics covered today are the 4 subsections of 8.4, the reading that was due today.
2. Give students 15 minutes to prepare their presentation and generate quiz questions for tomorrow's class.
 - a. Use a timer and announce time at 10- and 5- minutes remaining so students can pace themselves.
 - b. Answers to commonly asked questions, and tips for the different groups may be found below:

1.5.2.1 ASSIGNMENT FAQ

1.5.2.1.1 Encapsulation & Abstraction

1.5.2.1.1.1 Guidance for student teachers

1. If students are lacking direction, encourage students to refer to the textbook. A good example of why it's important to encapsulate code can be found in the "Did you know" blue box in section 8.4.
2. A good "Tricky Code Cheat Sheet" tip would be for students to form a rule their peers can memorize for when its' appropriate to encapsulate code.

1.5.2.1.1.2 Common Questions/Answers

1. Why can't we just ignore code, or promise not to modify it instead of encapsulating it?

Most large programs are authored by many people over the course of months or years! Sometimes you don't need (or want) to know the details of another programming segment—if it ain't broke, don't fix it. To keep from accidentally changing code, it's best practice to "protect" it by encapsulation.

1.5.2.1.2 Private Fields

1.5.2.1.2.1 Guidance for student teachers

1. If students are having trouble outlining their lesson, suggest that they start with the fourth complete version of code in the textbook. If they use the textbook example, make sure they point out which program is the client code.
2. A good "Tricky Code Cheat Sheet" tip would be the convention of fields at the top of the class, followed by constructors, followed by methods.

1.5.2.1.2.2 Common Questions/Answers

1. What is the scope of private fields?

Private fields are visible to all of the code inside the Point class, including the instances of the class. Client code cannot directly refer to an object's fields if you've encapsulated them (marked them as private).

2. But what if we still want client code to have to read some of the fields?

Write an accessor method (a "get" method). It returns a copy of the field's values to the client, so the client can see the values, but can't modify them.

1.5.2.1.3 Class Invariants

1.5.2.1.3.1 Guidance for student teachers

1. Remind students to include the definition of a class invariant for students to record in their notebooks.
2. Since we did not cover the this keyword, you should point out to the group that the this keyword in the example allows the code to directly refer to the implicit parameter. Since this might be confusing, you should give this group extra assistance, or assign your most advanced group this lesson.
3. Using division/modulus 60 to convert seconds to minutes or minutes to hours as outlined in the book is a useful tip for the "Tricky Code Cheat Sheet."

1.5.2.1.3.2 Common Questions/Answers

1. How is enforcing a class invariant different from using a class constant?

A class constant will always stay the same value; it does not ever change. A class invariant is a fact about some data that you as the programmer assert will always be true. For instance, in Pokemon, each stat must be less than 255—that is the max value you can have for any one stat. The value itself may go up or down depending on what happens to your Pokemon in the game, but it will always be less than 255.

2. Can you use a mutator method to change the class invariants?

No, so pick and choose your invariants with caution!

3. Skip the exceptions example—exceptions are not on the AP exam and will not be tested!

1.5.2.1.4 Changing Internal Implementations

1.5.2.1.4.1 Guidance for student teachers

1. If students need guidance on structuring their lesson, encourage them to coordinate with the group teaching class invariants, since they will be modifying the code taught by that group. They might also want to refer to the “Did You Know” blue box in the section.
2. A good tip for the Tricky Code Cheat Sheet could be the simpler time span code.

1.5.2.1.4.2 Common Questions/Answers

1. How come changing the internal design does affect client code?

Since you encapsulated your class, the client code won’t know that you changed the internal state. The only way your changing the class code can mess with a client program is if you change the constructors or method headers.

1.5.3 Student Preparation of Lesson [30 minutes]

1. Give each group 7 minutes to present their topic and 2 minutes for questions.
2. Encourage students to ask questions, and be sure to ask a question or two of each team (depending on how many teams you have).
3. Use the grading rubric as outlined here:

3 pts.	2 pts.	1 pts.	0 pts.
Presentation includes definitions and an example with proper syntax.	Presentation includes definitions or an example with proper syntax.	Presentation includes definitions or an example with proper syntax with few mistakes.	Presentation includes definitions or an example with proper syntax with many mistakes.
Presentation includes a non-example as helpful contrast.	Presentation includes a non-example that is marginally helpful.	Presentation includes a non-example that does not add to comprehension.	Presentation includes a non-example that adds confusion, or presentation does not include a non-example.
Presentation includes a helpful tip that is clearly explained and concisely stated.	Presentation includes a helpful tip that is clearly explained or concisely stated.	Presentation includes a helpful tip that is not clearly explained and may include a small error.	Presentation does not include a helpful tip or hint.

1.6 College Board Topic Questions

After this lesson, students will be able to answer questions from the College Board Unit 5 Topic Questions 5.4 Accessor Methods, 5.5 Mutator Methods, 5.6. Writing Methods and 5.8 Scope and Access.

1.7 Accommodation and Differentiation

Circle around the room to help students through reading the text in the textbook. Make sure that each of your working teams is properly stratified (rather than using tiered grouping).

If students are speeding along, encourage students to write down questions to pose to other groups during mini-lessons. If a group finishes early, encourage them to rehearse lesson delivery.

1.8 Teacher Prior CS Knowledge

Java is inconsistent with its use of encapsulation which could cause confusion with beginning students. When getting the length of a String `a`, the access method `.length()` is used. However, when getting the length of an array, the public, though final, instance variable `.length` is used.

1.9 Misconceptions

Students get the notion that because of data encapsulation, classes cannot be fields of another class. Even though the data in the class is protected from being modified directly by other classes, the class itself can be used as fields in other classes.

1.10 Videos

- BJP 8-4, *Encapsulation*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c8-4
- CSE 142, *Encapsulation* (35:38–45:58)
https://www.youtube.com/watch?v=V3Gs1Ug82_E&start=2138
- CSE 142, “*this*” notation (optional) (45:59–49:46)
https://www.youtube.com/watch?v=V3Gs1Ug82_E&start=2759

1.11 Forum discussion

Lesson 5.04 Encapsulation (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 5.05 – Finding & Fixing Errors

1.1 Overview

1.1.1 Objectives – *Students will be able to...*

- **Find** errors in their returned homework assignments.
- **Correct** their code

1.1.2 Assessments – *Students will...*

- **Re-submit** all homework assignments with corrected answers.

1.1.3 Homework – *Students will...*

- **Review** materials for the Picture lab by:
 - **Reviewing** all the blue pages at the end of Chapter 8
- **Submit** 5 questions for review in class tomorrow using electronic survey
- **Update** all summaries in notebook for the upcoming notebook check

1.2 Materials & Prep

- **Any student homework assignments** that you have not yet returned
- **Student self-help system** (such as C2B4 or student pairing)
- **Electronic survey** for student review requests

The homework tonight asks students to submit 5 questions for review. Create an electronic survey for students to complete with 6 text fields, one for name, and 5 for questions they have about Ch. 8 content. Set a deadline by which time students must have submitted 5 questions from Ch.8 that they would like to see reviewed after completion of the Pictures lab. If students do not have questions, stipulate that they still have to submit something to receive credit, even if it is only questions they think other students may have.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction and homework distribution	5min
Student work	35min
Students trade work, check, and submit	10min

1.4 Procedure

Today we continue reinforcing concepts and applying the tools, procedures, and code that were introduced last week. Students will have the opportunity to correct any incorrect homework or classwork assignments. If students did not have time to finish the homework from yesterday, you may allow them time to work on that today.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and Homework Distribution [5 minutes]

1. Return student homework packets, or have students place their returned homeworks in a pile on their desk.
2. Explain to students that they have the opportunity to get full credit on their homework grades by correcting them now, in class. Ask students for suggestions/ideas on how to make sure they don't miss any errors.

By now students should be used to relying on their error checklist/algorithm.

1.4.3 Student Work [35 minutes]

Have students work individually to correct their homework grades.

- Offer time checks for students so they stay on task.
- If students have not finished homework assignments, allow them time today to complete these assignments to turn in for partial credit.

1.4.4 Students trade work, check, and turn in [10 minutes]

At the end of class, have students trade their homework assignments to evaluate each other's corrections before submission.

1.5 Accommodation and Differentiation

In ELL classrooms, pair students and allow them to work together to correct their work. If you noticed a particular problem was difficult for the majority of students, read the question aloud and help students work through it.

For those students who have nothing to correct (or finish very early), reward them with silent free time, or allow them to work on a free-choice programming project.

1.6 Forum discussion

Lesson 5.05 Finding & Fixing Errors (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 5.06 — Picture Lab

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Complete** a long-form lab, using two-dimensional arrays of objects, array traversing algorithms, program analysis, binary numbers, and inheritance.

1.1.2 Assessments — *Students will...*

- **Complete** the Picture Lab

1.1.3 Homework — *Students will...*

- A list of homework assignments is provided below.

1.2 Materials & Prep

- **Projector and computer**
- **Picture Lab Teacher's Guide**
- **Classroom copies** of the Picture Lab Student Guide
- **Associated Picture Lab & Picture Lab Extension Files**
- **Digital camera (optional)**
- **CD (optional)**
- **Egg cartons and small candies** (Skittles or M&Ms) (**optional**)
- **Photo negative (optional)**
- **Rectangular mirror (optional)**

Read through the Teacher, Student, and Extension guides ahead of time to familiarize yourself with the parts of this long-form lab. Using the guides, complete the lab on your own to spot possible challenges for your students. Upload all student files onto each computer desktop for student access. Don't give the finalClasses folder to your students—it contains sample answers!

NOTE: If your students enter the classroom with prior programming knowledge, or if your class is moving through the AP course quickly with ease, you may want to deliver the Text Excel lab (included in Unit 5 materials) instead.

1.3 Pacing Guide: Day 1

Section	Total Time
Student Activity 1 & 2	Full class
Homework: <i>Summarize your notes in your notebooks. Notebook checks in class tomorrow!</i>	TONIGHT

1.4 Pacing Guide: Day 2

Section	Total Time
Student Activity 3 & 4	Full class
Notebook Checks	During class
Homework: <i>Outline Chapter 8</i>	TONIGHT

1.5 Pacing Guide: Day 3

Section	Total Time
Student Activity 5	Full class
Notebook Checks	During class
Homework: <i>Read and highlight Chapter 2 of Barron's review book. Skip "The this Keyword".</i>	TONIGHT

1.6 Pacing Guide: Day 4

Section	Total Time
Student Activity 5 & 6	Full class
Notebook Checks	During class
Homework: <i>Take the Chapter 2 exam in Barron's review book, skipping #20. Grade your answers.</i>	TONIGHT

1.7 Pacing Guide: Day 5

Section	Total Time
Student Activity 6, continued	Full class
Check Barron's review books for highlighting note-taking, and practice test completion and correction	During class
Homework: <i>Read and highlight Chapter 5 of Barron's review book.</i>	TONIGHT

1.8 Pacing Guide: Day 6

Section	Total Time
Student Activity 7	Full class
Homework: <i>Read BJP 8.5 and answer self-check questions 29–30</i>	TONIGHT

1.9 Pacing Guide: Day 7

Section	Total Time
Student Activity 8	Full class
Creating a collage	During class
Homework: <i>Finish up creating a collage</i>	TONIGHT

1.10 Pacing Guide: Day 8

Section	Total Time
Student Activity 9	Full class
Simple edge detection algorithm and implementation	During class
Homework: <i>Continue working on Simple edge detection.</i>	TONIGHT

1.11 Pacing Guide: Day 9

Section	Total Time
Student Activity 9, continued	Full class
Finish Simple edge detection	During class
Homework: <i>Submit 5 review questions on the electronic survey.</i>	TONIGHT

1.12 About Barron's

- Barron's is an AP CS A review book that some school provide students. If your school doesn't provide Barron's there are many alternative homework assignments that can be found at <http://codingbat.com/java> (no classes).
- Alternatively, you can save time spent on the lab by checking activities as Homework.

1.13 Procedure

All guides, sample code, answer code, and example code may be found in the folder "Milestone 2 Picture Lab."

1. To help students start the lab smoothly, start Activity 1 as a whole group.
2. Encourage students to use their Tricky Code Cheat Sheets, 4 Commandments of Scope, notebooks, textbooks, classroom posters, and homework assignments.
3. Offer occasional time-checks to help keep students on pace.
4. Grade notebooks and review books in between helping students so students can keep notebooks for homework and studying in the evenings.

1.14 Accommodation and Differentiation

In ELL classrooms, read all directions aloud before breaking into individual practice, and allow up to twice the amount of time for completion of the lab.

- To save time on the rest of the lab, don't spend too much time reviewing binary numbers, and restrict color exploration (Activity 2) to ~20 minutes.
- Use the tactile exercises as suggested on page 6 of the Teacher's guide (candy exercise and exploring the digital camera).

As needed, allow students to pair up to help each other with reading comprehension (but remind students that they each must submit their own code). Each day that you begin the lab, start with a quick survey of student concerns and questions.

- Adaptations for group work can be found on page 19 of the Teacher's guide.

Assessment questions have been relocated to the practice exam, WS 5.7.

1.15 Forum discussion

[Lesson 5.06 Picture Lab \(TEALS Discourse account required\)](#)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 5.06 — Data Lab

1.1 Overview

1.1.1 College Board 2019 Labs

In the summer of 2019, the college board released 4 new labs. Either the Picture Lab or Data Lab can be assigned after the completion of Unit 5.

1.1.2 Objectives — *Students will be able to...*

- **Complete** a long-form lab, using two dimensional arrays of objects, array traversing algorithms, program analysis, binary numbers, and inheritance.

1.1.3 Assessments — *Students will...*

- **Complete** the College Board's AP CS A Data Lab.
Students will answer end of activity Check your understanding and open-ended activity.

1.1.4 Homework — *Students will...*

- **A list of homework assignments is provided below.**

1.2 Materials & Prep

- **Projector and computer**
- **Data Lab** Teacher's Guide
- **Classroom copies** of the Data Lab Student Guide
- **Associated Data Lab & Data Lab Extension** Files

Read through the Teacher, Student, and Extension guides ahead of time to familiarize yourself with the parts of this long-form lab. Using the guides, complete the lab on your own to spot possible challenges for your students. Upload all student files onto each computer desktop for student access. Don't give the finalClasses folder to your students—it contains sample answers!

1.3 Pacing Guide: Day 1

Section	Total Time
Student Activity 1	Full class
Homework: <i>Summarize your notes in your notebooks. Notebook checks in class tomorrow!</i>	TONIGHT

1.4 Pacing Guide: Day 2

Section	Total Time
Student Activity 2	Full class
Notebook Checks	During class
Homework: <i>Outline Chapter 8</i>	TONIGHT

1.5 Pacing Guide: Day 3

Section	Total Time
Student Activity 3	Full class
Notebook Checks	During class
Homework: <i>Read and highlight Chapter 2 of Barron's review book. Skip "The this Keyword".</i>	TONIGHT

1.6 Pacing Guide: Day 4

Section	Total Time
Student Activity 3 (day 2)	Full class
Notebook Checks	During class
Homework: <i>Take the Chapter 2 exam in Barron's review book, skipping #20. Grade your answers.</i>	TONIGHT

1.7 Pacing Guide: Day 5

Section	Total Time
Student Activity 4	Full class
Check Barron's review books for highlighting note-taking, and practice test completion and correction	During class
Homework: <i>Read and highlight Chapter 5 of Barron's review book.</i>	TONIGHT

1.8 Pacing Guide: Day 6

Section	Total Time
Student Activity 4 (day 2)	Full class
Homework: <i>Read BJP 8.5 and answer self-check questions 29–30</i>	TONIGHT

1.9 Pacing Guide: Day 7

Section	Total Time
Student Activity 4 (day 3)	Full class
Creating a collage	During class
Homework: <i>Finish up creating a collage</i>	TONIGHT

1.10 Pacing Guide: Day 8

Section	Total Time
Student Activity 4 (day 4)	Full class
Simple edge detection algorithm and implementation	During class
Homework: <i>Continue working on Simple edge detection.</i>	TONIGHT

1.11 Procedure

All guides, sample code, answer code, and example code may be found by logging into the College Board AP Audit side and downloading the Steganography Lab materials.

1.11.1 About Barron's

- Barron's is an AP CS A review book that some schools provide students. If your school doesn't provide Barron's there are many alternative homework assignments that can be found at codingbat.com/java
- Alternatively, you can save time spent on the lab by checking activities as homework.

1.11.2 General Project Notes

1. To help students start the lab smoothly, start Activity 1 as a whole group.
2. Encourage students to use their Tricky Code Cheat Sheets, 4 Commandments of Scope, notebooks, textbooks, classroom posters, and homework assignments.
3. Offer occasional time-checks to help keep students on pace.
4. Grade notebooks and review books in between helping students so students can keep notebooks for homework and studying in the evenings.

1.12 Accommodation and Differentiation

In ELL classrooms, read all directions aloud before breaking into individual practice, and allow up to twice the amount of time for completion of the lab.

- To save time on the rest of the lab, don't spend too much time reviewing binary numbers, and restrict color exploration (Activity 2) to ~20 minutes.
- Use the tactile exercises as suggested on page 6 of the Teacher's guide (candy exercise and exploring the digital camera).

As needed, allow students to pair up to help each other with reading comprehension (but remind students that they each must submit their own code). Each day that you begin the lab, start with a quick survey of student concerns and questions.

Assessment questions have been relocated to the practice exam, WS 5.7.

1.13 Forum discussion

Lesson 5.06a Data Lab (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 5.07 — Review

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Identify** weaknesses in their Unit 5 knowledge.

1.1.2 Assessments — *Students will...*

- **Create** a personalized list of review topics to guide tonight's study session.

1.1.3 Homework — *Students will...*

- **Study** for tomorrow's test using targeted review list

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and marker**
- **Results** from electronic survey of review topics
- **Classroom copies** of the practice test [WS 5.7](#)

Once students have submitted their review requests, assemble those topics into categories and prepare to re-teach the topics as needed. If you are crunched for time, hold the review session during the second half of Day 12 of the Picture Lab.

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Review of student questions	30min
Sample test review	15min
Check student study lists	5min

1.4 Procedure

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Review of Student Questions [30 minutes]

1. Clearly indicate that you expect all students to have a list of review topics to study this evening. Periodically remind students that this list will be checked at the end of class.
2. Begin with a review of student-submitted questions before reviewing the practice questions.

1.4.3 Sample Test Review [15 minutes]

1. Begin review with practice test WS 5.7. Have students work through section I questions, then review the answers as a class.
2. Give students time to complete section II questions, then review the answers as a class.
3. Finally, work through the various review topics, prioritizing questions that popped up the most.
 - a. Some questions you may address while working through the sample test.
 - b. Be ready for additional questions to pop up as you go. Save yourself the work and use old homework questions and student-generated test questions as examples to work through.
4. Use a combination of group-solving questions on the whiteboard, think-pair-share, and timed-response as review strategies.
5. After you've completed reviewing an idea, remind the class that they should write down that topic if they feel they still have to review it tonight.

1.4.4 Check Student Study Lists [5 minutes]

Spend the last 5 minutes of class checking each student's review topic list.

1.5 Forum discussion

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 6.00 — Test Review & Reteach

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Re-learn or strengthen** content knowledge and skills from Unit 5.

1.1.2 Assessments — *Students will...*

- **Re-submit** test answers with updated corrections for partial or full credit
 - Credit depends on instructor preference

1.1.3 Homework — *Students will...*

- **Read** BJP 9.1
- **Correct** any incorrect test answers by re-answering on a separate sheet of paper
 - To get back credit, they must justify their new answers
 - Staple new answer sheet to old test and turn in tomorrow

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Corrected student tests**
- **Student grades** (posted online, emailed to students, or handed back on paper in class)
- **Digital copy of test questions** for projector

1.3 Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Class discussion (if needed)	10min
Test review and reteach	35min
Check student notes and return tests	5min

1.4 Procedure

Return student grades before class begins or while students are completing the bellwork.

Do not return students' tests before the review session, since you want to motivate students to pay attention to the entire review, taking supplemental notes the entire time.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Class Discussion (if needed) [10 minutes]

1. If grades are low, invite the class to a discussion of what can be improved. Begin with student complaints and suggestions to build student buy-in. Ask students:
 - how they felt they were going to do before the test
 - what surprised them once they were taking the test
 - what they felt worked in the first unit (lessons, review strategies, assignments)
 - what do they think they want to change for the second unit
2. Once you feel that a dialogue has been established, validate students' feelings, then challenge them (e.g. AP courses are stressful, but this is good practice for college, where the pace is faster and professors don't give personalized instruction).

1.4.3 Test Review and Reteach [30 minutes]

1. Walk the students through each question on the test, glossing over questions that everyone answered correctly.
 - a. You can ask for students to volunteer answers, or call on students randomly. Make sure that students explain their logic when they answer. If a student gives an incorrect answer, the explanation will tell you what you need to re-teach or clarify.
 - b. Do not skip questions that everyone answered correctly, but do not spend more than the time it takes to read the question, and congratulate students' correct answers.
2. Project a copy of each question as you review—this will help students recall the question/process the information.
3. Make sure that students are taking notes during the re-teach, reminding students that for homework, they will have an opportunity to win back some of the points on their exam.
4. For Section II questions, select a sample of student work (with any identifying information obscured), and work through the answer together as a class.

1.4.4 Check student notes and return tests [5 minutes]

At the end of class, check student notes, and return the tests in hard copy form if applicable.

1.5 Accommodation and Differentiation

Encourage advanced students to take on additional programming challenges. One easy way to do this is to assign Programming Projects from the blue pages at the end of each Chapter.

If you have a few students that are struggling with the class, choose these students to create your classroom posters after school or for extra credit.

1.6 Forum discussion

Lesson 6.00 Test Review & Reteach (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 5.XX — Programming Project (PictureLab Alternative)

1.0.1 Overview

This lesson is intended to be a smaller scale alternative (2-3 hours) to PictureLab from the original AP computer science course. Starting from 2014-2015 the College Board announced that 20 hours of hands-on lab time may replace the original required labs (MagPie, PictureLab, Elevens). In this lesson document, we recommend some open-ended alternatives.

1.0.2 Objectives — *Students will be able to...*

- **Conduct user-centred research** to identify specific functions for a specialized application
- **Plan and create** a calculator that perform specialized operations for an end-user
- **Test, evaluate, and share** the end product

1.0.3 Assessments — *Students will...*

- **Apply** 2-dimentional arrays, traversal, binary representations of data (images, characters)
- **Submit** a complete, functional program

1.0.4 Pacing Suggestion

- The duration of this project is at the discretion of the teacher. We recommend 2 hours for class time design and implementation, and 1-2 hours for out of class time to connect with end-user.
- Project involves conducting research work (survey or interviews), and communicating with end-user, out of the classroom.

1.0.5 Implementation Details

1.0.6 Complexity and Creativity

Students should come up with the idea themselves, based on user-centred research, and ideate a specialized application to address the needs of a specific user group.

Students will follow applied design process to implement the idea. You should talk to your teacher often to ensure that your progress is in-line with expectations.

1.0.7 Documentation and Style

1.1 - As with all projects, your program must be well-written, well-documented, and readable. Writing code with good style is always good idea. This will help you debug, pick up where you left off each day, and keep track of progress.

1.1.1 STEP 1 - UNDERSTANDING CONTEXT

Conduct user-centred research to find design opportunities and barriers. Select an end-user for whom you will design and create this program (this can be a friend, classmate, relative, etc). Create interview questions that will allow you to understand the end-user's interests and likes/dislikes. Here are some possible applications that could potentially use "if" statements and "String" methods:

- Picture Manipulation** * One of the Activities in the original Picture Lab (estimated 1-2 hour time)
- * Create an image manipulation for a special purpose

Other Canvas-based Applications, involving ARRAYS, Binary representations of data * Etch-a-Sketch https://en.wikipedia.org/wiki/Etch_A_Sketch * Battleship [https://en.wikipedia.org/wiki/Battleship_\(game\)](https://en.wikipedia.org/wiki/Battleship_(game)) * Othello or Tic-Tac-Toe with a twist * Maze creation for a nanobug to run through * Hex dump Translator * Flip Book Creator (create series of images where only a few pictures differ) https://en.wikipedia.org/wiki/Flip_book * Snowflakes, or Flower drawing program

1.2 * Interactive story scene

- After you have some initial ideas, interview some potential end-user to clarify your project specifications.

1.2.1 Emphasize with students...

1.2.2 Big Ideas - personal design interests require the evalution and refinement of skills

The possibilities are endless.

We live in a world that operates on the written or spoken word. The computer and internet has forced us to digitize the words we use into zero's and one's for transmission purposes. But at a higher level, most people are working with "strings" (a collection of character symbols, making up a word). Sometimes words are easily understood and recognized, sometimes it is not.

Our human brains naturally make decisions and choices. With so many options, and paths, we can ask the computer to help us explore these choices. This is great when it comes to routine, repetitive, or time consuming tasks. Poets, writers, travellers, wanderers, cryptographers, kings and queens have loved playing with words (strings, characters), and creating games (logic, secret messages) since the earliest of times to entertain, inspire, and protect the land.

Let your creative juices flow!

1.2.3 STEP 2 - DEFINING AND IDEATING

Choose a design opportunity and point of view, make inferences about limitations and boundaries. Take creative risks to identify gaps to explore, generate a range of possibilities, prioritize ideas for prototyping. Using the responses from your end-user interview, begin to develop a plan for your custom application. Given the duration of the project, you should limit your calculator to do an interesting activity for a specific end-user. At

this point you will only be be to implement a text-based user interface (ie, no graphical elements, like buttons or scrollbars). Keep it simple and easy to use.

List:

- * the types of calculator functions it will do:
- what will the user input be?
- what output will be?
- * what user interaction with the program will the user have?
- how do you start the activity?
- how does the activity end?

Share these ideas with your end user. Record their comments, suggestions and feedback and note any changes that you may make as a result of this interview. Identify any issues or problems that might arise as you begin to program (what areas might require more information or programming solutions? Where can you find this information?)

1.2.4 CHECKPOINT 1

SUBMIT YOUR INTERVIEW QUESTIONS, RESPONSES AND PROGRAM DESIGN PLAN TO YOUR TEACHER.

1.3

1.3.1 STEP 3 – PROTOTYPING AND TESTING

Construct prototypes, making changes to code as needed. Program your calculator. Be sure to review course notes and activities to make sure that you effectively implement object oriented programming design for your application. When you are ready, create a short test plan and test your program. Include expected output and actual output and include details related to any fixes that needed to be made. A good motto to remember is **code a little, test a little**. Have your end-user try a working version of your program. Note their suggestions, comments and feedback. Indicate any changes that you made to the program, based on the user's feedback.

1.3.2 CHECKPOINT 2

SUBMIT THE CURRENT VERSION OF YOUR PROGAM TO YOUR TEACHER, AS WELL AS YOUR TEST PLAN AND FEEDBACK FROM THE END-USER

1.3.3 STEP 4 – SHARING, TESTING AND FINAL ITERATION

Gather feedback from users over time to critically evaluate your design and make changes to product design or processes. Identify new design issues.

Share your work with other classmates, friends or family. Record any feedback, suggestions and comments and use this information to make the final iteration of your program.

1.3.4 CHECKPOINT 3

SUBMIT THE FINAL VERSION OF YOUR PROGAM TO YOUR TEACHER, AS WELL AS YOUR FEEDBACK FROM CLASSMATES, FRIENDS OR FAMILY.

Grading Scheme/Rubric

Implementation	
Project is appropriately complex and creative	4 points
Program is well-documented and shows good style	10 points
Final product meets all requirements and goals laid out in checkpoint specifications	8 points
Program uses programming concepts effectively, including all required elements with an appropriate level of complexity	4 points
Object Oriented Programming concepts are effectively applied with an appropriate level of complexity	6 points

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 6.01 — Inheritance Basics

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Correctly define** inheritance.
- **Use** proper syntax to extend a class.
- **Illustrate** is-a relationships.
- **Properly implement** constructors of derived classes using super.

1.1.2 Assessments — *Students will...*

- **Complete** a Class Hierarchy poster as indicated in WS 6.1

1.1.3 Homework — *Students will...*

- **Read** BJP 9.2 up to “DividendStock Behavior.”
- **Collect** images that represent instances of the classes created for in-class poster project.

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Classroom copies** of [WS 6.1] Start class poster, [Example 6.1]
- **Pictures** of Pokémons (<http://tinyurl.com/l6mybmr>) or Pokémons Cards (optional)
- **Video** about bees (https://www.youtube.com/watch?v=K3oMN1a_pdg)
- **Student pair assignments**
- **Art supplies** for each group:
 - Poster paper or cut sheets of butcher paper
 - Lined paper (at least 3 sheets per group)
 - Markers
 - Glue Sticks
 - Old magazines, flyers, newspapers to cut up for collage
 - Scissors
 - Yarn, string, or embroidery floss
 - Tape, magnets, or tacks to hang finished work

Most of the supplies required for this lesson are readily available in high schools. If the school doesn't have poster paper, butcher paper works well. Other supplies may be available to borrow from the Math, Science, or Art teachers. To get an idea what a final student project should look like, check out the picture of sample student work “Example 6.1.”

1.3 Pacing Guide: Day 1

Section	Total Time
Bell-work and attendance	5min
Introduction	20min
Review of the project	5min
Student work	25min

1.4 Pacing Guide: Day 2

Section	Total Time
Student work & teacher check	20min
Peer review	10min
Whole-group discussion and reteach if needed	15min
Quiz	5min

1.5 Procedure

Hook your students by prominently displaying art materials and sample work (of your own making, or saved from a previous year). To feature the most engaging student examples, look for work that has many instances of each class and uses classes/objects that are popular with your class. Invoke an air of mystery and don't offer an explanation for any of it.

1.5.1 Bell-work and Attendance [5 minutes]

1.5.2 Introduction [20 minutes]

1.5.2.1 Emphasize with students...

1.5.2.2 Big Ideas - Products can be designed for life cycle

This activity will take you through the process of designing superclasses and subclasses. As you do so, consider carefully how your superclass could be reused and repurposed, later, by another programmer to create something different.

Many of the programs you create could be altered, remixed and tweaked to create new and innovative software that is slightly different than the original. This is a great thing about Object Oriented Programming. Much of the code can be reused and repurposed to create things we haven't even thought of yet.

1. Have a quick class discussion about bees with the students jotting down some notes on the board.
2. Watch the video about bees and ask again what they now know adding to the notes.
3. Ask them to help you create a Bee class with attributes (ex: boolean carryingPolen) and methods (ex: fly(), gatherNectar()). This does not need to be compilable code — a UML class diagram would work well here.
4. They may come up with this on their own, but steer them in the direction of multiple types of bees — worker, queen, drone (male). Create another UML box, but don't connect them yet.
5. Ask what all bees have in common. Write everything down even if it doesn't have a perfect programming analogue (e.g. yellow stripes) or if it isn't something that all bees do (e.g. sting or gather nectar).
6. Start explaining the idea of superclasses and subclasses. Explain that subclasses are specialized versions of the superclasss. Suggest that the Bee class could be the superclass.
7. Ask students what the specialized subclasses would be. Start mapping out the heirarchy with arrows. Explain that the arrow represents an "is a" relationship. Give other examples of "is a" relationships

1.5.2.3 Examples

- An apple tree is a tree.
 - A stegasaurus is a dinosaur.
 - An electric Pokémon is a Pokémon.
 - A computer science student is a student.
 - A math teacher is a teacher.
 - Soda is a drink.
 - A square is a rectangle.
 - A smart phone is a computer.
8. Have students give examples of other things have have this relationship and ask them to point out the superclasses and subclasses. Allow examples to have multiple subclasses.
 9. Ask students to define an inheritance hierarchy in their own words. Briefly discuss why you would want to use inheritance in programming.
 - An **inheritance hierarchy** is a set of hierarchical relationships between classes of objects.
 - **Inheritance** is a programming technique that allows a derived class to extend the functionality of a base class, inheriting all of its state and behavior.)
 - **Superclass** is the parent class in an inheritance relationship.
 - **Subclass, or child class** is the derived class in an inheritance relationship.
 10. Check for understanding by returning to the examples above, and asking students to give an example of some characteristics (fields) the parent class would have, and what characteristics students would add to the specialized subclasses.

Example: Drinks could have a String *name* and boolean *carbonated*, and Soda could add a boolean *caffeinated*.

11. The class header for a subclass that extends the functionality of the parent class looks like this:

```
public class Mammal extends Animal {  
public class Motorcycle extends Vehicle {  
public class Churro extends Pastry {
```



- Point out that the subclass names are capitalized by convention, and that you always use the `extends` keyword. Give students a moment to think of a few hierarchical relationships in a think-pair-share, and ask several volunteers to come to the front of the room to demonstrate the correct class header.
12. For the following example, we will create subclasses that extend the `Vehicle` superclass written below. Show the slide with this code and/or create a new file with the code below. If you think your students can come up with the methods relatively quickly, you can show portions of the code and have them fill in the methods in pairs.

```

class Vehicle {
    static final int FEET_IN_MILE = 5280;
    private int xCord;
    private int yCord;

    private double fuelGallons;
    private double maxFuel;
    private double mileFuelRatio;

    public Vehicle(int xStart, int yStart,  double fuelGallons, double maxFuel, double milePerGallon){
        this.xCord = xStart;
        this.yCord = yStart;

        this.fuelGallons = fuelGallons;
        this.maxFuel = maxFuel;
        this.milePerGallon = milePerGallon;
    }

    public void move(int dx, int dy){
        // if there is enough fuel, move dx and dy feet.
        double distance = Math.sqrt(Math.pow(dx, 2) + Math.pow(dy, 2));
        double fuelUsed = distance / (FEET_IN_MILE * milePerGallon);

        if (fuelGallons - fuelUsed > 0){
            this.xCord += dx;
            this.yCord += dy;
            this.fuelGallons -= fuelUsed;
        } else {
            System.out.println("Not enough fuel.");
        }
    }

    public void refuel(double fuel){
        this.fuelGallons += fuel;
        if (this.fuelGallons > this.maxFuel){
            this.fuelGallons = this.maxFuel;
        }
    }

    public int getX(){
        return xCord;
    }

    public int getY(){
        return yCord;
    }

    public double getFuel(){
        return fuelGallons;
    }
}

```



13. Explain that this is a superclass and you wouldn't necessarily instantiate something as simply a vehicle. Ask for examples of subclasses. Write down all examples — even if someone says airplane right away — but tell the class that to keep it simple you'll start with a couple land-based examples like car and truck. Here are some examples that are in the slide. Feel free to hide that slide and make these classes with the students. Make sure you are repeating OOP vocab like inheritance, "is a", superclass, subclass, constructors, etc.

```

class Car extends Vehicle {

    public Car(int xStart, int yStart){

```

```

        super(xStart, yStart, 15, 15, 20);
    }
}

``` Java class Semi extends Vehicle {
public Semi(int xStart, int yStart){ super(xStart, yStart, 130, 130, 6.5); }
}

``` Java
class Motorcycle extends Vehicle {
public Motorcycle(int xStart, int yStart){
    super(xStart, yStart, 5, 5, 50);
}
}

```

- //
- //
14. When you have a couple land-based examples down, ask students to create a vehicle subclass and constructor that calls the superclass. Good examples could be Tractor, PickupTruck, SportsCar, SUV, Tank. Advanced students can try something that doesn't have fuel like Bicycle, Skateboard, HorseDrawnCarriage.
 15. Ask them to share their classes with a neighbor, and optionally share with the class.
 16. Bring them back together and ask how they would start an Airplane class. Ask them what attributes and methods they will need in the Airplane class that the Vehicle class doesn't have. Here is an example that is also in the slide deck:

```

``` Java class Airplane extends Vehicle {

private int zCord; private boolean landingGear;

public void liftoff(int dx, int dy, int dz){ if (zCord > 0){ System.out.println("We're already flying!"); return; }

// if there is enough fuel, move dx, dy, and dz feet. double distance = Math.sqrt(Math.pow(dx, 2) + Math.pow(dy, 2) + Math.pow(dz, 2)); double fuelUsed = distance / (5280 * milePerGallon);

if (fuelGallons - fuelUsed > 0){ xCord += dx; yCord += dy; zCord += dz; fuelGallons -= fuelUsed; landingGear = false; } else{ System.out.println("Not enough fuel."); } }
```

```

Your students may point out different attributes and methods — write down ideas as they come and allow other students to edit them to function better. Emphasize that designing classes is not an exact science and there are lots of right answers. Explain that they should expect to have to restructure their classes as they iterate through the design process.

If you have time, ask the students to write the constructor for Airplane in pairs.

Note: some students may notice we are accessing private attributes in Airplane (xCord and yCord), but explain that we will have some fixes for that later.

1.5.3 Review of the Project [5 minutes]

1. Briefly review the assignment with your students, reading the directions aloud if need be.
2. If you haven't already distributed project materials at this point, do so while your students are rearranging into partner pairs.

1.5.4 Student Work [25 minutes]

1. Encourage students to take 5-10 minutes on Step 1. They should review all steps of the project to ensure that their selection of classes lends itself to the project (e.g. they shouldn't pick something they don't know a lot about because they'll have trouble coming up with fields and methods).

- Offer time checks every 10 minutes so students can stay on pace. By the end of the first day, they should have gotten to step 6 or 7. Visit each group to make sure that they haven't veered off course.
- On **day two**, check student work and help students display their work around the room.
- Check that the flow-of-control string (see WS 6.1 for explanation) correctly shows how a method is passed through subclasses to the superclass.
- Remind students to take notes (Step 11 on WS 6.1) to help them remember talking points for later in the class.
- As a whole group, ask students to volunteer what they really liked about others' projects. Solicit questions and critiques, re-teaching if needed.
- Administer quiz 6.1 to assess student understanding.

1.6 Accommodation and Differentiation

Encourage advanced students to add additional classes, fields, methods, and client code. If students still have time to spare, encourage them to read on method overriding, and invite them to add that code as well. Students can attach this "extra code" using paper and tape/glue or sticky notes.

If you have a few students that are struggling with the class, give them your starter Vehicle class and some subclasses, and let them build off of your examples. You can print out your starter code and cut it into pieces and shuffle them so students have to place each line in the correct location (as with a Parson problem).

If your students need further instruction on calling a superclass' constructor, go through a few more examples using the Vehicle superclass, or classes that you created as a whole group.

1.7 Teacher Prior CS Knowledge

- The Object Oriented Programming (OOP) paradigm could be thought of as mimicking the real world where objects consists of data that define them and actions that can be performed on the data. As you will see, the process of learning OOP is infinitely more complex.
- The pillars of Object Oriented Programming (OOP): inheritance, encapsulation, and polymorphism. In a nutshell inheritance allows for code reuse by defining methods once in a superclass, encapsulation provides data security by hiding data implementation from the user and only allowing methods in the class to modify the data, and polymorphism offers flexibility to the designer by way of methods defined in many forms.

1.8 Misconceptions

Students' use of "inheritance" prior to computer science are in the context of inheritance from an ancestor and genetic inheritance of traits. However, in computer science, inheritance is used for classification where the class that inherits (extends) from a more general class (super class). Neither of the students' prior knowledge and use of inheritance is an accurate representation of Java's class structure.

1.9 Common Mistakes

Object oriented concepts common mistakes:
<http://interactivepython.org/runestone/static/javareview/oobasics/oomistakes.html>

1.10 Video

- BJP 9-1, *Inheritance: Interacting with the Superclass*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c9-1
- CSE 142, *Inheritance* (23:28–35:06)
<https://www.youtube.com/watch?v=WPdv8X291hE&start=1408>
- CS Homework Bytes, *Inheritance, with Zach*
https://www.youtube.com/watch?v=Alv2ApK_jdo

1.11 Forum discussion

Lesson 6.01 Inheritance Basics (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 6.02 — Overriding Methods & Accessing Inherited Code

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Replace** superclass behavior by writing overriding methods in the subclass.
- **Write** subclass methods that access superclass methods.

1.1.2 Assessments — *Students will...*

- **Add** code to their Class Posters from the previous lesson.

1.1.3 Homework — *Students will...*

- **Read** the rest of BJP 9.2 starting from “The Object Class.”

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Classroom copies** of [WS 6.2](#)
- **Class posters** from 6.1
- **Art supplies** for each group:
 - Markers
 - Poster paper (alternatively printer paper and tape)
 - Tape, magnets, or tacks to hang finished work

1.3 Pacing Guide

| Section | Total Time |
|--------------------------------------|------------|
| Bell-work and attendance | 5min |
| Introduction & review of the project | 10min |
| Student work | 15min |
| Peer review | 10min |
| Whole-group discussion/critique | 10min |

1.4 Procedure

Hook your students by prominently displaying art materials, and sample work (of your own making, or saved from a previous year). To feature the most engaging student examples, look for work that has a particularly complex or impressive flow-of-control pattern (see WS. 6.2 for explanation). If you have the time, this would be a great opportunity to display some complex code for students to examine on their own time. The more intricate the string pattern on the poster, the more intriguing the sample will appear.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction & Review of the Project [10 minutes]

1.4.2.1 Emphasize with students...

1.4.2.2 Big Ideas - Products can be designed for life cycle

Software organizations are always updating and altering software source code. This is done in order to fix bugs, add features or improve performance. It is for this reason that it's always important to remember that software can be designed for life cycle.

The original programmer can ensure that their code is well written and well documented. This can facilitate later updates and changes.

The original programmer can also think carefully about how additional features and updates might be implemented. This can impact the original design of objects or classes in order to ensure the ability to further add to and improve the design.

1. Inheritance makes it convenient to reuse code between classes. However, sometimes we'll want to specialize code in a subclass, or ignore a method that doesn't apply. Ask students for examples when this might be the case. If they're having trouble thinking of concrete examples, ask them to think of an example from their own class hierarchy that they created in the previous lesson. Some examples to get the class started:

- Subclass *Mammal* might have a special case of Animal superclass method *feedYoung* (because they lactate).

- Subclass *HotDrink* might use a different method *consume* from the *Drink* superclass (maybe the method involves sipping or burning your tongue).
 - Subclass *Drone* will have to have a modified *sting* method that doesn't actually sting.
2. Replacing superclass behavior by writing a new version of the methods in the subclass is called **overriding**. To override a method, write the method you want to replace in the subclass! No special syntax is required!

- Building on our *Vehicle* example from the last lesson, we can write our own *move* methods for subclass *Submarine*:

```
class Submarine extends Vehicle {

    private int zCord;

    public void move(int dx, int dy){
        // Nuclear subs don't really run out of fuel so no need to check.
        this.xCord += dx;
        this.yCord += dy;
    }

    public void move(int dx, int dy, int dz){
        this.xCord += dx;
        this.yCord += dy;
        this.zCord += dz;
    }
}
```



- Compare this to the *Vehicle* superclass method *move*, which is reproduced here for convenience:

```
public void move(int dx, int dy){
    // if there is enough fuel, move dx and dy feet.
    double distance = Math.sqrt(Math.pow(dx, 2) + Math.pow(dy, 2));
    double fuelUsed = distance / (5280 * milePerGallon);

    if (fuelGallons - fuelUsed > 0){
        this.xCord += dx;
        this.yCord += dy;
        this.fuelGallons -= fuelUsed;
    }
    else{
        System.out.println("Not enough fuel.");
    }
}
```



3. Have students point out the differences between the two methods, predict the new output, and offer additional or alternative changes to the overridden *Submarine move* method.

4. In this example, *HybridCar* uses the battery if it is charged, otherwise it reverts to the same move method the *Car* and therefore *Vehicle* class use.

```
class HybridCar extends Car {
    private double batteryLife;
    private double batteryPerMile;

    public void move(int dx, int dy){
        // If the battery is charged use it, otherwise use the fuel (use super.move())
        double distance = Math.sqrt(Math.pow(dx, 2) + Math.pow(dy, 2));
        double batteryUsed = distance / (5280 * batteryPerMile);
        if (batteryLife - batteryUsed > 0){
            batteryLife -= batteryUsed;
        }
        else{
            super.move(dx, dy);
        }
    }
}
```



- Ask students why it's valid to call the overridden method *move*, and reference the superclass method *move* by the same name. (The superclass method is accessed using dot notation, which tells Java where to direct the flow of control.)

5. Pause at this point to point out that you have seen vehicles that in real life use several different kinds of engines (4 stroke internal combustion in cars, diesel in semis, turbines in airplanes, nuclear in subs, electric in hybrids), but your move methods do not need to worry about that. We have *abstracted* out the internal workings and are only focused on the input (fuel) and the output (movement in the coordinate space).

5. Ask if they notice what is wrong with the code as it is (accessing private attributes `xCord` and `yCord` — something we've done in a few classes so far so maybe they'll have noticed it then). What if we want to access other information directly from the *Vehicle* class? Remember, our class had fields for `x`, `y`, and `fuelGallons`, but they're all private because we were smart and remembered to encapsulate them.

- If we wanted to write a method in our *HybridCar* subclass that accesses the data contained in `xCord`, we would have to add a *get* method first in the Superclass *Vehicle*:

```
public int getX(){  
    return xCord;  
}
```



- This makes a copy of `xCord` that is public, and can be accessed outside of the *Vehicle* class.

6. If we want to change the value in a private attribute, we could use a setter that would check to make sure the value is valid.

7. Have students offer additional examples using the *Vehicle* superclass, or using examples from their own class hierarchy.

8. Complete the introduction by asking students to explain what the difference is between overriding and overloading methods. (Overloading methods is when one class contains multiple methods with the same name, but a different number of parameters—sometimes called the *parameter signature*.)

1.4.3 Student Work [15 minutes]

1. Briefly review WS 6.2 with your students, reading the directions aloud if need be.

2. If you haven't already distributed project materials at this point, do so while your students are rearranging into partner pairs.

3. Encourage students to take 5 – 10 minutes on Step 1. They should review all steps of the project to ensure that their additional methods make sense.

Announce that you'll offer extra credit to funny or creative code (if that fits in with your teaching style).

4. Offer time checks so students can stay on pace. Before you allow students to begin the peer review tour of others' work, remind them to take notes on their feedback so they will be able to contribute to the group critique/discussion at the end of class.

1.4.4 Peer Review [10 minutes]

Allow students 10 minutes to tour each other's work and offer feedback.

1.4.5 Whole-group Discussion/Critique [10 minutes]

If possible, rearrange student seats into a circle for the critique to encourage informal discussion.

- As a whole group, ask students to volunteer what they really liked about others' projects.
- Solicit questions and critiques, and re-teach if needed.
- Award classroom participation points to all students who contribute to the discussion.

1.5 Accommodation and Differentiation

Encourage advanced students to add additional classes, fields, methods, and client code. If students still have time to spare, encourage them to increase code complexity, add additional levels to the class hierarchy, or help their peers.

If you have a few students that are struggling with the assignment, allow them to work in groups of 4, each pair helping the other with their code. If students need additional guidance, have students complete the worksheet as a series of think-pair-shares, where you return to whole group to share and discuss answers before moving on to the next step. **Teaching the class this way will roughly double the time required to complete the exercise.**

1.6 Misconceptions

When first learning polymorphism, students learn method override before method overload. However, in order to successfully override a method, the subclass method must have the same method signature as the superclass, otherwise the method will be overloaded. The code must match the method parameters and return type and the methods public, they cannot be private or static. When helping students debug their code where the overridden method is not behaving as anticipated, asking the student if the method signatures match can help find the error on their own.

1.7 Video

- BJP 9-2, *Polymorphism*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c9-2
- CSE 142, *Polymorphism* (35:07–49:57)
<https://www.youtube.com/watch?v=WPdv8X291hE&start=2107>

1.8 Forum discussion

Lesson 6.02 Overriding Methods & Accessing Inherited Code (TEALS Discourse account required)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 6.03 — Interacting with the Object Superclass

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Replace** superclass behavior by writing overriding methods in the subclass.
- **Write** subclass methods that access superclass methods.

1.1.2 Assessments — *Students will...*

- **Complete** Practice questions
- **Complete** a worksheet

1.1.3 Homework — *Students will...*

- **Read** BJP 9.3 up to “Interpreting Inheritance Code.”

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Classroom copies** of [WS 6.3](#), [Poster 6.3](#)
- **Poster 6.3**
- **Empty & washed, or non-refrigerated, drink bottles**, with labels affixed (**optional**)
- Teacher access to CS Awesome [Unit 3 Lesson 07 Comparing Objects Lesson Plan](#) Sign up at [CS Awesome AP CSA Java Curriculum](#)
- Access to Dr. Nguyen [Compound Boolean Expressions and Comparing Objects](#) slide deck
- Access to CS Awesome [3.7. Comparing Objects](#)

Poster 6.3 is set to print a movie-sized poster of 15" x 20". If you do not want to print a poster this size, access the .pptx version of the poster, and reset the page size to legal, ledger, or whatever larger-format paper you have available to you. Note: Some fonts on this poster print incredibly small.

Arrange the drink bottles in a space where students can pick them up and look at them if need be. Encourage students to look at the bottles to get ideas for fields they can use in their *Drink* subclasses.

1.3 Pacing Guide

| Section | Total Time |
|--------------------------|------------|
| Bell-work and attendance | 5min |
| Review of the project | 10min |
| Student practice | 15min |
| Student practice: WS 6.3 | 25min |

1.4 Procedure

Most of student practice today is a review and further integration of the concepts that were introduced since the beginning of this unit. The only new concept being drilled today is the `equals` method.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Review of the Project [10 minutes]

1. All classes are subclasses of the `Object` class. Whether you write `extends` in the header or not, all classes inherit the `Object` class. It is built into Java this way, so you never have to explicitly write `extends Object` in a class header.
 - o This means that all code inherits some generalized methods that come automatically with the `Object` class.
2. The AP exam covers `toString` and `equals` methods only.
 - o Ask students if they remember what `toString` method does. (It gives the class name followed by a location in memory, which isn't very helpful, so we always create our own `toString` methods when we create a class.)
 - o Ask students if they remember why we can't just use `==` to test for equality between objects.
3. The `==` operator tests whether two objects have the same identity, if they refer to the same object, not whether the two objects are in the same state.

```
String z = "z";
String a = z + z;
String b = "zz";

a == b;           // Evaluates to false because a and b refer to different Strings

String c = b;
c == b;           // Evaluates to true because c and b refer to the same String
```

4. The default `equals` method (that comes with your `Object` superclass) does the same thing as the `==` operator. (Since the `equals` method comes from the `Object` superclass, it interprets its input parameter as an object.) But for Strings, the `equals` method does something smarter:

```
a.equals(b);    // Evaluates to true because the content of a and b are the same "zz"
c.equals(b);    // Evaluates to true because c and b refer to the same String
```

- o To rewrite an `equals` method that compares object state (to override the `Object` version of the `equals` method), you need to cast the object in order to let Java know that the objects really can be compared.

- To test if two *Drink* objects have the same name and serving size, you would write an *equals* method that looks like this:

```
public boolean equals(Object o) {  
    Drink other = (Drink) o;  
    return name.equals(other.name) && (ounces == other.ounces);  
}
```



5. Comparing Objects with == and !=

- See CS Awesome [Unit 3 Lesson 07 Comparing Objects Lesson Plan](#)
- Slides 22-25 of [Compound Boolean Expressions and Comparing Objects](#) can be used to introduce this topic
- Have students navigate to CS Awesome [3.7. Comparing Objects](#), view the Activity 1 video and complete Activities 1-7.

1.4.3 Student Practice: [15 minutes]

1. Have students work individually or in pairs to complete the following Practice questions:

- a. Self-Check 9.3: subclassSyntax
- b. Self-Check 9.10: inheritanceVariableSyntax
- c. Self-Check 9.8: CarTruck
- d. Self-Check 9.9: CarTruck2

1.4.4 Student Practice: WS 6.3 [25 minutes]

1.4.4.1 Emphasize with students...

1.4.4.2 Content - Problem decomposition - Advanced programming structures - Management of complexity

Now that you are creating superclasses and subclasses you are starting to program using quite advanced programming structures. These structures are sometimes difficult concepts for students to grasp, but once understood, they can actually facilitate programming.

The creation of superclasses and subclasses allows you to break your program down into smaller chunks (this is called decomposition). It also allows you to manage complexity by having methods and attributes exist within the class. This keeps the data separate from the main program, but of course it can be accessed or altered whenever necessary.

Once students have completed these exercises, distribute worksheet 6.3.

- Read through the questions aloud, if needed.
- If you are having the students work in IDE, be sure to review your procedure for submitting work electronically before students begin.
- Encourage students to explore the drink bottles to get ideas for fields they can use in their *Drink* subclasses.

1.5 Accommodation and Differentiation

At this point of the course, introducing TextExcel may be beneficial. Although the students don't have the required knowledge to complete the project (yet), TextExcel and can be useful when explaining polymorphism in the following lesson because of how cells are displayed. If you do decide to go this route, it's a good idea to give your class at least a half-day to work on TextExcel to get a better understanding of the prompt and where they get stuck in the project.

Encourage advanced students to add additional classes, fields, methods, and client code. If students still have time to spare, encourage them to increase code complexity, add additional levels to the class hierarchy, or help their peers.

If you have a few students that are struggling with the assignment, allow them to work in groups of 4, each pair helping the other with their code. If students need additional guidance, have students complete the worksheet as a series of think-pair-shares, where you return to whole group to share and discuss answers before moving on to the next step. **Teaching the class this way will roughly double the time required to complete the exercise.**

1.6 Misconceptions

Students often have confusion on the difference between overriding vs overloading methods. The following is a chart of the differences:

| | Overriding
Method with same name,
parameters, and return
type | Overload
Methods with same name but
parameters and/or return
type |
|-----------------------------|--|--|
| Method
Signature | Same | Different |
| Class
hierarchy | Subclass overrides superclass | Overloaded method can exist anywhere in class hierarchy |
| Behavior | Change behavior of superclass method | Multiple behavior |
| Execution | Run Time | Compile Time |

1.7 Video

- CSE 143, *Interacting with Objects* (2:59–30:39)
<https://www.youtube.com/watch?v=ewsM5Iak83g&start=179>
- CSE 143, *Interacting walkthrough* (31:38–40:10)
<https://www.youtube.com/watch?v=ewsM5Iak83g&start=1900>

1.8 Forum discussion

Lesson 6.03 Interacting with the Object Superclass (TEALS Discourse account required)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 6.05 – Has-a Relationships

1.1 Overview

1.1.1 Objectives – *Students will be able to...*

- **Identify and explain** why two classes have an is-a or a has-a relationship.
- **Create** a has-a relationship between two classes

1.1.2 Assessments – *Students will...*

- **Complete** an AP Section II question “Trio”

1.1.3 Homework – *Students will...*

- **Read** BJP 9.5 (Optional if covering Interfaces)

1.2 Materials & Prep

- Day 1
- **Projector and computer**
- **Whiteboard and markers**
- **Projection or classroom copies** of [WS 6.5](#) (an AP Section II question)
- **Student pair assignments**
- **Video for hook:** <https://vimeo.com/18439821>
- Day 2
- Teacher access to CS Awesome [**Unit 9 Lesson 5 Inheritance Hierarchies Lesson Plan**] Sign up at [CS Awesome AP CSA Java Curriculum]
- Access to CS Awesome [**Lesson 9.5. Inheritance Hierarchies**]

1.3 Pacing Guide Day 1

| Section | Total Time |
|----------------------------------|------------|
| Bell-work and attendance | 5min |
| Introduction (with pair work) | 15min |
| Student practice | 25min |
| Student share/whole group review | 10min |

1.4 Pacing Guide Day 2

| Section | Total Time |
|---|------------|
| Bell-work and attendance | 5min |
| CS Awesome 9.5. Inheritance Hierarchies | 35min |

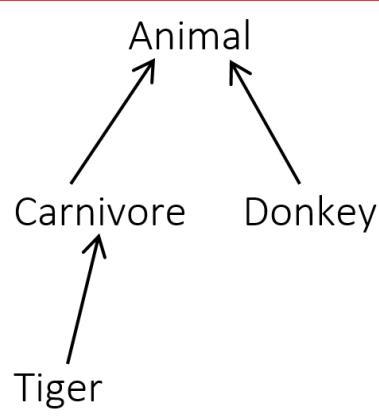
1.5 Procedure Day 1

Hook your class today with the Shapeshifter video (link in “Materials & Prep”). Ask students what the different animals had in common in the video, and what real animals have in common. What behaviors did the animals have in common? What was a behavior that stuck out as unique to each animal? The idea here is to get students thinking about how to use inheritance (and eventually polymorphism) to reuse code for different classes.

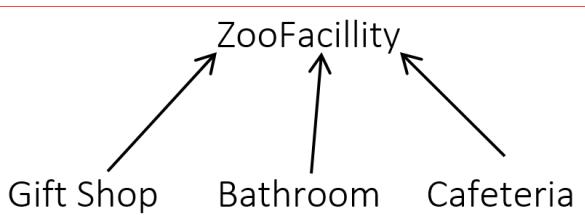
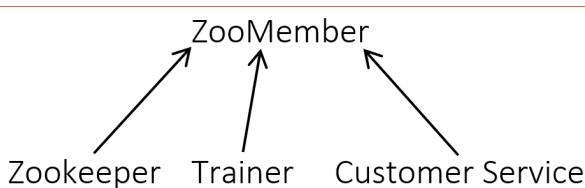
1.5.1 Bell-work and Attendance [5 minutes]

1.5.2 Introduction (with pair work) [15 minutes]

1. Continue the class discussion by prompting students to pair up and draw out a class hierarchy including: *Animal*, *Carnivore*, *Tiger*, and *Donkey*.
2. After students have created the hierarchy, ask them to consider some behaviors that would also apply to the classes *Zoo* and *Zookeeper*. Students should share these behaviors and jot them down in their notebooks for later reference.
3. Using student-generated examples, illustrate the *is-a* relationships, and areas where you might include *has-a* relationships:
 - o An ***is-a*** describes inheritance between a subclass and a superclass. The superclass inherits all the code from the superclass because you can think of it as a subset of the superclass. (Hopefully a student can volunteer an answer like this.)



- Have students create class hierarchies for ZooStaff and ZooFacilities. (Some staff titles could be zookeeper, customer service representative, or trainer. Typical zoo facilities include bathrooms, gift shops, and cafeterias.)



- Has-a** describes the relationship between a class that is client code of another class. The class is a component of the “client class,” and one object *contains* the other. To create a *has-a* relationship, write a field in the class that refers to the other class:

```

public class Zoo {
    private Animal [] animals;
    private ZooMember [] zooMembers;
    private ZooFacility [] zooFacilities;
    ...
  
```

- We create fields that refer to other classes (highlighted) to create a *has-a* relationship. When you can't substitute one class for another, you should use a has-a relationship. A zoo is not an animal (or array of animals), nor is it a staff member (or array of staff members). A zoo contains, or *has*, all of these components.

1.5.3 Student Practice [25 minutes]

- Students should remain in their student pairs. If your classroom has computers, students should complete the following exercise and you should review the protocol for submitting assignments electronically.
- Before students get started, ask students what types of meals they see in fast food restaurants, and what options are typically bundled together. Guide students to discuss:
 - Sandwiches: chicken, burger, fish
 - Drinks: soda, water, juice, coffee, milk

3. Sides: French fries, onion rings, tatertots, salad, apple slices
3. In the examples here, ChickenSandwich, BurgerSandwich, FishSandwich, SodaDrink (etc.) are subclasses. A very rich class hierarchy can be formed here! Explain to students that they're going to generate complex code that uses inheritance and polymorphism to model the state and behavior of *ValueMeals*.
4. On the projector, whiteboard, or as a handout, give students the 2014 AP exam Section II question 4 problem "Trio." A copy of this problem has been included in your materials as WS 6.5.
5. Give students ≈20 minutes to write and debug their sample code. Walk around the room checking on students and offering guidance if they are stuck or on the wrong path. Choose one or two student pairs to share their code with the class during whole-group discussion, and help those students save and transfer their files to the projecting computer.

1.5.4 Student Share/Whole Group Review [10 minutes]

In a whole group, ask for students to share their code and explain how the established the *has-a* relationship between classes. Check for student understanding by asking why a *has-a* relationship was more appropriate than creating a **is-a* (*hierarchical) relationship.

1.6 Procedure Day 2

1.6.1 Bell-work and Attendance \[5 minutes\]

1.6.2 CS Awesome Lesson 9.5. Inheritance Hierarchies \[35 minutes\]

The student lesson uses CS Awesomes [Lesson 9.5. Inheritance Hierarchies]. There you will find the lesson plan and activities to check for student understanding. The teacher lesson plans are accessible by joining the [Teaching CSAwesome google group] and navigating to [Unit 9 Lesson 5 Inheritance Hierarchies Lesson Plan]. Students should complete activities 1-6.

1.7 Accommodation and Differentiation

1.7.1 Common Questions/Issues

- You may have to remind students to write `toString` methods in the other classes as well. Have students demonstrate flow of control on the `toString` method to demonstrate why a `toString` method in the "client class" will require a `toString` method in the other classes.
- Check to make sure students are using private fields. If you're seeing many students using public fields, pause the class to lead a whole group discussion about why encapsulation is so important. (The Y2K bug is a particularly exciting example of what happens when we don't encapsulate code. Be sure to describe the panic and doomsday predictions that this caused!)
- If your students have been having trouble tracing the flow of control as a method calls another method in a different class, take some time to demonstrate how to use the step-into and step-over buttons in the IDE.
 - The step-over and step-into buttons can be found in the top toolbar.
 - In the screenshot below, the pointer rests on the step-into button, which will advance the flow of control one step at a time. The step-over button will jump to the end location of control once the

entire method has executed.



If students complete this assignment quickly, encourage them to increase the complexity and depth of the program. Some ideas to get students started:

- Introduce an Aquarium class with associated *has-a* and *is-a* relationships to some of the classes you already created.
- Create additional subclasses in Animal, or classes such as AquariumStaff as needed.

If you'd like for students to expand on the AP question given, have students:

- Create additional subclasses for Drink/Sandwich/Salad (if not already done)
- Create a Menu of ValueMeal options, and allow user to select the options they want to create a meal
- Add serving sizes to Drink and Side, and create a SuperSizeValueMeal
- Add price to Sandwich/Drink/Salad and have ValueMeal prices be 85% of the total item prices.

If students are struggling with the assignment, allow more time (up to two class periods) to complete the lesson. Read the prompt aloud for the class, and do the steps together if needed. In classes with ELLs, you can distribute saved .java files that contain an entire functional program, with shuffled components. Students will need to organize the code fragments into the proper order (a Parsons problem).

1.8 Teacher Prior CS Knowledge

Is-a relationships define class hierarchy through inheritance, while *has-a* relationships define classes through the incorporation of component classes.

1.9 Misconceptions

Has-a implies a one-to-one relationship, while in many cases, classes have a one-to-many relationship. Use examples where the *has-a* (composition) relationship is more than just a 1:1 relationship. For example, a deck of cards has 52 cards.

1.10 Forum discussion

[Lesson 6.05 Has-a Relationships \(TEALS Discourse account required\)](#)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 6.06 — Interfaces (Optional)

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Implement and use** interfaces.

1.1.2 Assessments — *Students will...*

- **Complete** an in-class competition

1.1.3 Homework — *Students will...*

- **Read** BJP 9.6 (Optional if covering Interfaces)
- **Summarize notes** in notebook for tomorrow's notebook check
- **For extra credit:**
 - Generate your own class hierarchy that demonstrates the same concepts illustrated by the Financial Class Hierarchy outlined in the book. The extra-credit project is due [one week from assignment]().

1.2 Materials & Prep

- **Projector and computer**
- **White paper and markers**
- **In Class Poster 6.6**
- **Small group assignments** (≈4 people per group)

1.3 Pacing Guide

| Section | Total Time |
|--|------------|
| Bell-work and attendance | 5min |
| Introduction | 15min |
| Class competition (small groups) | 20min |
| Whole group review and competition judging/award | 15min |

1.4 Note

Interfaces was removed from the AP CS A exam starting in 2019-20. This lesson is therefore options. Only teach this lesson if you feel you have sufficient time to cover the remaining topics before the exam. Each class is different so use your judgement in estimating the time remaining.

1.5 Procedure

Hook your class by announcing a class competition. Feature TEALS swag, extra credit points, free homework passes, or raffle tickets as prizes. Break students into their small groups before beginning your introduction.

1.5.1 Bell-work and Attendance [5 minutes]

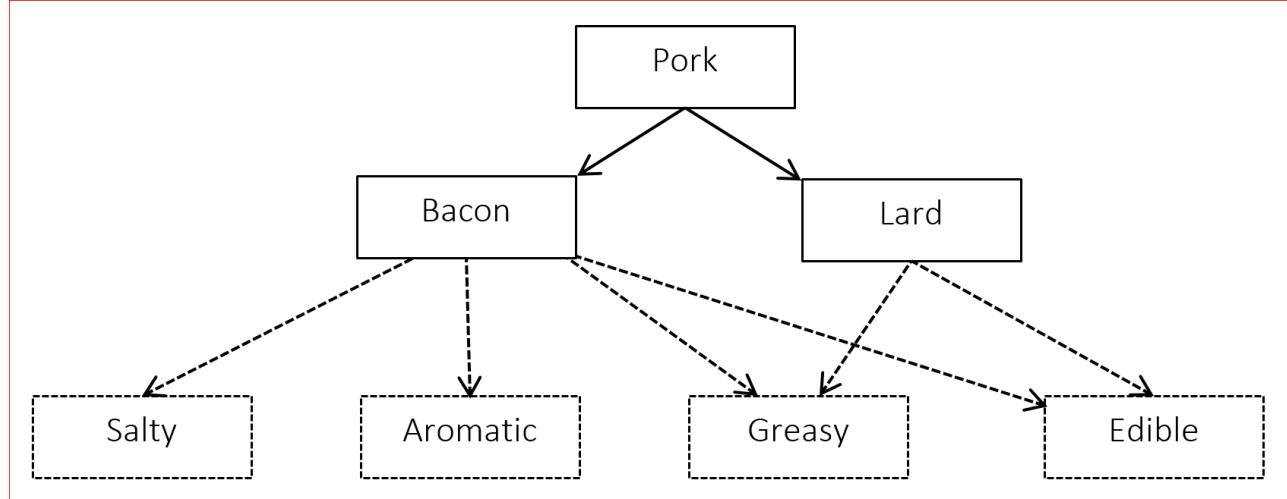
1.5.2 Introduction [10 minutes]

1. Begin with a lecture/discussion about inheritance and interfaces.
2. We've seen that inheritance is a useful tool for programming. (Ask students why inheritance is useful: it enables polymorphism and code sharing.)

But inheritance has limitations:

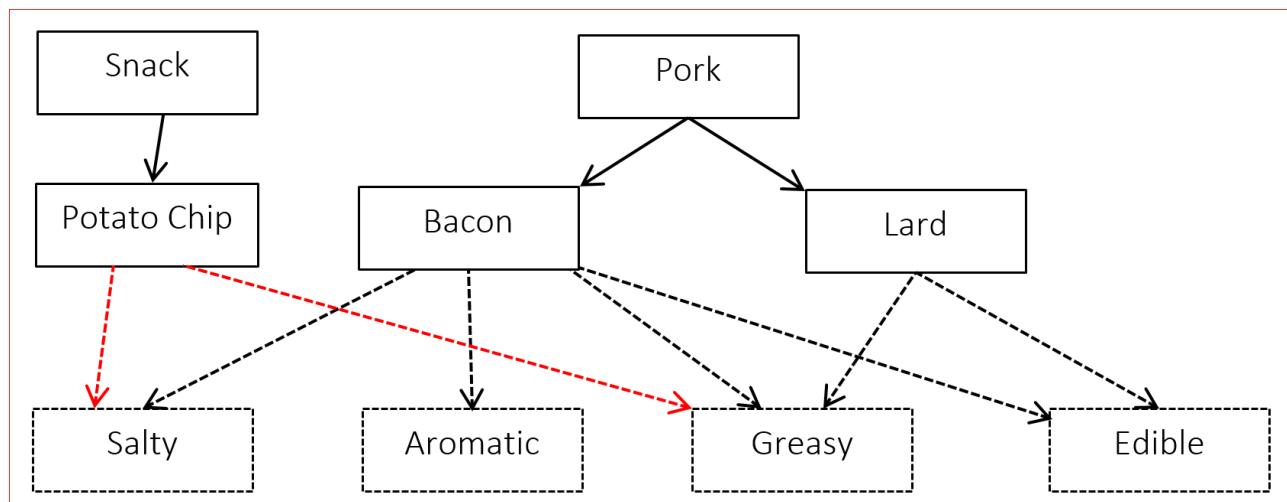
- Ask students if we can inherit code from more than one superclass. (*No.*)
 - If you want an is-a relationship or polymorphism, but you don't want to give a subclass access to the code, inheritance won't give you the encapsulation you need.
3. There's a special tool we can use called an *interface* that allows you to represent a common *supertype* between classes without actually sharing code.
 - An **interface** consists of a set of method declarations without a method body.
 - Think of the interface as a *promise* of behavior; the method is declared, but not defined. The method *will be* (is promised to be) defined in the subclass method.

If you don't define the method in your subclass (if you don't follow through on your promise), you'll get an error.
 4. Demonstrate what this looks like in practice. You can use the example given below, or ask the class to help you generate an example of their own choosing. Start with a sketch so students can visualize the relationships between interfaces and classes, and review inheritance vocabulary as you construct the diagram:



Interfaces are represented by dotted boxes and arrows, class and class hierarchies are represented by solid lines and boxes.

If you wish to show your students that interfaces can be shared by unrelated class hierarchies, you can add to the diagram above as shown below. The color of the interface arrows has been changed to red so it's easier to see where to draw the lines.



```

public interface Salty {
    double sodiumContent();
} // This is a promise to implement the sodiumContent
   // in the class that implements the Salty interface.

public interface Aromatic {
    String describeAroma(); // Notice the lack of "public"! Public is *assumed*.
}

public interface Greasy {
    double amountOfGreaseInMg(); // Point out that interfaces look just like classes
} // but without fields or method bodies

public interface Edible {
    double calories();
}
  
```

5. As you write the class header below, point out the keyword `implements` and match up the interfaces in the header with the interfaces in the diagram.

```

public class Bacon extends Pork implements Salty, Aromatic, Greasy, Edible {
    private double amountInKg;

    public Bacon(double amount) {
        amountInKg = amount;
    }
  
```

```
    }  
  
    public double calories() {  
        return amountInKg * CALORIES_PER_KG_OF_BACON;  
    }
```

- Ask students to point out the header and the constructor. Ask them if they can guess why you included a *calories* method. If they don't remember the answer from their reading assignment, point to the interfaces without any additional comment (you're following through on your *promise* to implement the method).
- Ask students if the *Bacon* class is complete. Have them help you fix it by adding the other methods you promised to implement in the *Aromatic*, *Greasy*, and *Salty* interfaces. Point out Poster 6.6 as an aid to help students write interfaces correctly.

1.5.3 Class Competition (Small Groups) [20 minutes]

1. On the board or overhead, project a series of five interfaces on a theme that you feel will creatively engage your class (you should blank out the parts that you feel will make the example most engaging). Some sample classes and interfaces with suggested methods:

1. Subclasses: Red, Orange, Yellow, Green, Blue
 - Interface: color
 - Sample Methods: double wavelengthInNm, Boolean isPrimaryColor
2. Subclasses: Wood, Brick, Adobe, Stone, Canvas
 - Interface: buildingMaterial
 - Sample Methods: String movableHousing, double costPerLb
3. Subclasses: Boeing 747, Pheasant, PaperAirplane, Cannonball
 - Interface: flies
 - Sample Methods: getMaximumAltitude, getRange, getSpeed

2. Invite each team to provide a team name, and explain the challenge:

1. Students should define classes to implement as many combinations of interfaces as possible.
 2. Students must include at least 1 method for each interface.
 3. Each class has to implement the interface methods and include a constructor.
3. The team with the most combinations of interfaces at the end of the time limit (\approx 20 minutes) win the competition.

1.5.4 Whole Group Review & Competition Judging/Award [15 minutes]

Leave time to have groups share their interfaces with the rest of the class during whole group discussion. Pay attention to any particularly creative or wacky examples that students come up with.

1.6 Accommodation and Differentiation

If your students require additional practice with interfaces before beginning the class competition, work through building a class *Lard* that implements the interfaces *Greasy* and *FoodItem*.

1.6.1 During the Activity

If your students require extra scaffolding, generate 1 method for each interface as a whole group before the competition begins. Write these methods on the board with the interfaces for easy reference.

If your students require an extra challenge, change rule c (in step 2 of *Activity*) to include additional methods and/or fields.

1.7 Teacher Prior CS Knowledge

A student may ask about the *Is-A* and *Has-A* relationships introduced in the previous lesson, which are great for describing inheritance vs composition. But what about interfaces? The relationship that can be used is *Can-Do*. A class that implements an interface can do the behaviors specified in the class it implements.

1.8 Misconceptions

Students often have confusion on the difference between extends vs implements. Syntactically is fairly straightforward to extend from a super class and implement from an interface. Conceptually, inheritance (extends) is used to define a class hierarchy where common functionality is factored out into the superclass. In contrast, interfaces (implements) is used to separate out the methods definition from the method implementation. No class hierarchy is implied when implementing an interface.

1.9 Video

- BJP 9-3, *Implement Comparable Interface*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c10-3
- CS Homework Bytes, *Interface and Abstract Classes, with Elizabeth*
https://www.youtube.com/watch?v=iiZ_TIZsE6Q
- CSE 143, *Interfaces* (note: uses ArrayList written in class as an example) (1:23–26:35)
<https://www.youtube.com/watch?v=KLJEss4jKf8&start=80>

1.10 Forum discussion

Lesson 6.06 Interfaces (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 7.00 — Test Review & Reteach

N.B. THIS LESSON IS OPTIONAL (See below for details.)

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Re-learn or strengthen** content knowledge and skills from Unit 6.

1.1.2 Assessments — *Students will...*

- **Re-submit** test answers with updated corrections for partial or full credit
 - Credit depends on instructor preference

1.1.3 Homework — *Students will...*

- **Read** BJP 13.1 up to "Sorting."
- **Correct** any incorrect test answers by re-answering on a separate sheet of paper
 - To get back credit, they must justify their new answers
 - Staple new answer sheet to old test and turn in tomorrow

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Corrected student tests**
- **Student grades** (posted online, emailed to students, or handed back on paper in class)
- **Digital copy of test questions** for projector

To accommodate for late-year scheduling drift, Unit 7 is planned with 4 periods of "wiggle room." Unit 7 includes a long-format lab that takes a minimum of 11 55-minute class periods to complete. **If you feel that you need to plan additional time for students to complete the Elevens lab, skip this review lesson, and assign test review and correction as a homework assignment.** Allow students to work together on the assignment so they can help each other with correcting answers. It is a good idea to check with each group before they leave class to ensure that they have A) exchanged contact information, B) have set up a location and time to do the assignment together.

The rest of the homework assignments can be shifted accordingly, since there are fewer homework assignments than there are days of the Elevens lab (see LP 7.3 for homework schedule).

1.3 Pacing Guide

| Section | Total Time |
|--------------------------------------|------------|
| Bell-work and attendance | 5min |
| Class discussion (if needed) | 10min |
| Test review and reteach | 35min |
| Check student notes and return tests | 5min |

1.4 Procedure

Return student grades before class begins or while students are completing the bellwork.

Do not return students' tests before the review session, since you want to motivate students to pay attention to the entire review, taking supplemental notes the entire time.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Class Discussion (if needed) [10 minutes]

1. If grades are low, invite the class to a discussion of what can be improved. Begin with student complaints and suggestions to build student buy-in. Ask students:
 - how they felt they were going to do before the test
 - what surprised them once they were taking the test
 - what they felt worked in the first unit (lessons, review strategies, assignments)
 - what do they think they want to change for the second unit
2. Once you feel that a dialogue has been established, validate students' feelings, then challenge them (e.g. AP courses are stressful, but this is good practice for college, where the pace is faster and professors don't give personalized instruction).

1.4.3 Test Review and Reteach [30 minutes]

1. Walk the students through each question on the test, glossing over questions that everyone answered correctly.
 1. You can ask for students to volunteer answers, or call on students randomly. Make sure that students explain their logic when they answer. If a student gives an incorrect answer, the explanation will tell you what you need to re-teach or clarify.
 2. Do not skip questions that everyone answered correctly, but do not spend more than the time it takes to read the question, and congratulate students' correct answers.
2. Project a copy of each question as you review—this will help students recall the question/process the information.
3. Make sure that students are taking notes during the re-teach, reminding students that for homework, they will have an opportunity to win back some of the points on their exam.

4. For Section II questions, select a sample of student work (with any identifying information obscured), and work through the answer together as a class.

1.4.4 Check student notes and return tests [5 minutes]

At the end of class, check student notes, and return the tests in hard copy form if applicable.

1.5 Accommodation and Differentiation

Encourage advanced students to take on additional programming challenges. One easy way to do this is to assign Programming Projects from the blue pages at the end of each Chapter.

If you have a few students that are struggling with the class, choose these students to create your classroom posters after school or for extra credit.

1.6 Forum discussion

[Lesson 7.00 Test Review & Reteach \(TEALS Discourse account required\)](#)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 7.01 – Searching Algorithms

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Compare and contrast** the different search algorithms.

1.1.2 Assessments — *Students will...*

- **Complete** some short-answer questions.

1.1.3 Homework — *Students will...*

- **Complete** self-check questions \#4-6 and exercises \#1-3

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Classroom copies** of [WS 7.1](#)
- **CS Unplugged Activity:** Searching Algorithms (<http://csunplugged.org/searching-algorithms>)
- **Classroom copies** of Battleship 1A, 1A', 1B, 1B', 2A, 2A', 2B, 2B'
 - Included in the CS Unplugged activity
 - **10-15 slips of paper** with different integers printed on them (1 integer per paper)
 - **Individually wrapped small candies**
 - **Student pair assignments**

1.3 Pacing Guide

| Section | Total Time |
|---|------------|
| Bell-work and attendance | 5min |
| Intro & demonstration | 10min |
| Student activity 1: Battleship Linear Searching | 15min |
| Student activity 2: Battleship Binary Searching | 15min |
| Worksheet completion/whole group discussion | 10min |

1.4 Procedure

Hook your students by placing the bowl of candy somewhere visible. Before the introductory lecture, announce that today's class is a "game day," and students will spend their time investigating computer algorithms by playing Battleship.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Intro & Demonstration [10 minutes]

1. Begin with a lecture/discussion about search algorithms.
2. Computers are useful because they can manage large collections of data quickly and easily. Ask students to give some examples about how large collections of information are managed by computers.

Examples: Data about items for sale are accessed as bar codes, schools store data about students, which can be accessed by student name, ID number, or grade level, weathermen store historical and current data about atmospheric conditions, etc.

3. Large collections of data aren't manageable unless we are able to search for a particular data point (datum).
- Example:* When you search the internet, you're searching for a single keyword (or phrase) called a "search key" within a particular webpage (or set of webpages).
4. Even though computers work very quickly, when we deal with searching large datasets, we need to use algorithms that are quick to use. A difference of a second or two is actually quite a lot when you think about how many times a day we use searches.
 - o Using your phone, the class clock, a watch, stopwatch, or your computer, demonstrate for the class what 3 seconds feels like. Ask students to imagine each websearch taking that long.
 - o Demonstrate 10 seconds, and ask students to imagine what would happen in a grocery store if each item scanned took 10 seconds for the price look-up. Ask for estimates on how long it would take a single family to check out groceries for the week, and have students offer predictions as to how this would affect business and consumer experience in the store.
5. When we decide as program designers which searching or sorting algorithms to use, we factor in:
 1. The size of the data array
 2. The space efficiency of the algorithm (how much memory it uses)
 3. Run-time efficiency (how fast it executes)

6. **Demonstration:** Using the CS Unplugged guide "Introductory Activity," get your students thinking about the process of and relationship between searching and sorting data. Use the introductory activity to introduce the Battleship games.

1.4.3 Student Activity 1: Battleship Linear Searching [15 minutes]

1.4.3.1 Emphasize with students...

1.4.3.2 Big Ideas - Tools and technologies can be adapted for specific purposes

When you first started looking at the linear and binary search algorithms, you probably didn't consider how they could be used in a game of battleship. This is what many find so interesting about Computer Science. The algorithms and solutions can be adapted to solve a wide variety of problems!

1. On the projector or the board, review the rules for Battleships – A Linear Searching Game from the CS Unplugged activity. Distribute sheets 1A and 1B to student pairs (face down so students don't see each other's papers).
2. Distribute WS 7.1 so students can answer questions as they play the Battleship games.
3. Give students ≈15 minutes to play the Battleship game and answer the corresponding questions on their worksheets. Students that complete the game with enough time to do a second round should receive 1A' and 1B'. Be sure to pace your students by announcing 5 minutes before transition.

1.4.4 Student Activity 2: Battleship Binary Searching [15 minutes]

1. Using the CS Unplugged guide for Battleships – A Binary Searching Game, explain the updated rules for this game. Distribute sheets 2A and 2B to student pairs (face down).
2. Remind students to answer the questions on their worksheets.
3. Give students ≈15 minutes to complete the game and answer their worksheets, then call the class together for a whole group discussion of their answers. Students that complete the game with enough time to do a second round should receive 2A' and 2B'.

1.4.5 Worksheet Completion/Whole-Group Discussion [10 minutes]

Discuss the worksheet questions as a class, assessing student understanding and re-teaching as needed.

1.5 Accommodation and Differentiation

To ensure that students understand the assignment, read the questions on WS 7.1 before students begin the activity.

Help students with metacognition by checking in with student pairs during the activities. Ask them to explain their decision making process to you, and if they are having trouble articulating their algorithms, ask them to explain one decision at a time.

Whenever possible, you should encourage students to do 2 rounds of Battleship for each search algorithm. This will allow students to track their own learning.

If your students are advancing through the course quickly and easily, you can augment this unit by having students write code to implement sequential or binary search. Assign further reading in the textbook (the latter $\frac{1}{2}$ of Chapter 13), and discuss with students how they can implement code that operate like the processes explored during class.

1.6 Teacher Prior CS Knowledge

- A binary search can be written using iteration, but binary search lends itself well to recursion. At this point in the curriculum, students have not learned recursion. You can revisit the binary search algorithm after introducing recursion.
- Big-O notation is not part of the AP CS A exam. However, some method of denoting relative size, whether it be time or memory usage, needs to be introduced. You can use big-O notation or invent one your own.
- The execution time of a linear search is n^2 while the execution time of binary search is $n \cdot \log_2(n)$. It is not necessary for students to fully understand logarithms in order to understand that binary search is faster than linear search for large values of n .

1.7 Video

- BJP 13-1, *Binary Search*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c13-1
- CSE 143, *Binary Search* (3:36–12:41)
https://www.youtube.com/watch?v=eCo_cxfKelU&start=216

1.8 Forum discussion

Lesson 7.01 Searching Algorithms (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 7.02 – Sorting Algorithms

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Compare and contrast** different sorting methods and **evaluate** their relative speed and efficiency.

1.1.2 Assessments — *Students will...*

- **Complete** some short answer questions on worksheets

1.1.3 Homework — *Students will...*

- **Read** BJP 13.1 “Sorting.”

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **CS Unplugged Activity:** Sorting Algorithms (<http://csunplugged.org/sorting-algorithms>)
- **Balance scales** (one for each group of students)
- **Classroom copies** of worksheet activities (included in CS Unplugged Activity)
- **Sets of 8 containers** of equal size but different weights (one for each group)
- **Student pair/group assignments**

Your physics, mathematics, or chemistry teacher may have weight sets and scales that you can use for this lesson. If you do not have weight sets, using prescription bottles, M&M Mini containers, or old film canisters filled with different amounts of sand will also work. If you do not have time to collect entire sets of these items, placing pennies, beans, or sand in sealed paper lunch bags will work in a pinch (warning: this can get messy if your students aren't careful!). It's important to choose opaque containers so students don't take the shortcut of eyeballing the weights instead of using the scale.

If you cannot find balance scales, directions on how to make a set can be found on these websites:

- <https://kriegerscience.wordpress.com/2011/09/28/how-to-make-a-set-of-weighing-scales>
- <http://www.wikihow.com/make-a-balance-scale-for-kids>

1.3 Pacing Guide: Day 1

| Section | Total Time |
|-------------------------------------|-------------------|
| Bell-work and attendance | 5min |
| Introduction | 10min |
| Student activity 1: Sorting Weights | 35min |

1.4 Pacing Guide: Day 2

| Section | Total Time |
|--|-------------------|
| Introduction & Setup | 5min |
| Student activity 2: Divide and Conquer | 30min |
| Whole group discussion & code demo | 10min |

1.5 Procedure

Hook your students by placing the materials for the lesson out around the room.

1.5.1 Bell-work and Attendance [5 minutes]

1.5.2 Introduction [10 minutes]

1. Lead a class discussion about why it would be important to sort data:
 - Ask students which searching method went faster yesterday (binary search), and what a prerequisite for this type of search was (array needs to be ordered/sorted).
 - Ask for additional examples of when it might be important to sort data. (Sorting songs in iTunes, sorting homework assignment files on your home computer, sorting goods in a store by location, price, or type, etc.)

1.5.3 Student Activity 1: Sorting Weights [35 minutes]

1. Distribute worksheets for “Sorting Weights” to student pairs or groups.
2. Review the directions with the class, and distribute materials.
 - If your class is not familiar with how to use the scales, take a few minutes to demonstrate the proper way to read the scales and record the weight data.
 - Be sure to remind students to count how many comparisons they make during the activity!
 - Make the “extra for experts” question mandatory.

3. Give students \approx 35 minutes to complete this activity—they will need to make 28 comparisons for this exercise.

1.6 DAY 2

1.6.1 Student Activity 2: Divide and Conquer [30 minutes]

1. Briefly review the next activity for the “Divide and Conquer” activity.
 - o Point out to students that they must try out the additional sorting methods on the last page (insertion sort or bubble sort).
 - o Remind students to keep track of how many comparisons they make for each type of sort.
2. Give students \sim 30 minutes to complete this activity, then call the class together for a whole group discussion of their answers.

1.6.2 Whole Group Discussion & Code Demonstration [10 minutes]

1. Discuss the worksheet questions as a class; ask students:
 - o what sorting algorithms were the quickest
 - o which sorting algorithms should use more memory
 - o which sorting algorithms they’ve used in their lives, and what those situations were. (Some examples to get them started include sorting notes or homework into date order, organizing shuffled cards, etc.)
2. The AP subset does not include specific sorting code, but students are required to conceptually understand the mechanisms used in each of the sorting algorithms. Students should be familiar with the best- and worst-case scenarios for each of the sorting algorithms. Give the students a summary of this information by having students copy the notes below, or by offering this information as a handout.

1.6.3 SORTING ALGORITHMS

1.6.3.1 Selection Sort

For an array of n elements, the array is sorted after $n-1$ passes. After the i th pass, the first i elements are in their sorted position.

The steps are:

1. The algorithm divides the input list into two parts: the sublist of items already sorted, which is built up from left to right at the front (left) of the list, and the sublist of items remaining to be sorted that occupy the rest of the list.
2. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list.
3. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist, exchanging it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

A useful animation of selection sort can be found on Wikipedia here: (<http://tinyurl.com/muo8ycd>).

1.6.3.2 Insertion Sort

For an array of i elements, the array is sorted after $n-1$ passes. After the i th pass, the first i elements are sorted with respect to each other, but not in their final sorted positions. The worst-case for insertion sort is when the array is initially sorted in reverse order (because sorting takes the most number of comparisons/moves). The best-case scenario is if the array is initially sorted in increasing order (because no elements will have to be moved).

The steps are:

1. Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list.
2. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there.
3. It repeats until no input elements remain.

A useful animation of insertion sort can be found on Wikipedia here: (<http://tinyurl.com/ldw8bj6>).

1.6.3.3 Comparison of Sorting Algorithms

Going in depth on every sorting algorithm is beyond the scope of this course. This is an introduction on comparing sorting algorithms.

1. Start a discussion on the insertion sort algorithm above on how would you measure the amount of resources used. Lead the students to:
execution time, memory, number of swaps.
2. What would the resources used be if the array was in reverse order for the selection sort vs insertion sort? What if the array was already sorted? What is the average execution time? Algorithms are classified by the worse case execution time. For both selection and insertion sort, they are n -squared algorithms.
3. Introduce the concept of non- n squared algorithms in the following video: <https://www.youtube.com/watch?v=ZZuD6iUe3Pc>. A faster sorting algorithm, Merge Sort, will be introduced in the recursion Unit.

1.6.4 College Board Topic Questions

After this lesson, students will be able to answer questions from the College Board Unit 7 Topic Questions 7.4 Developing Algorithms using ArrayList.

1.6.5 Accommodation and Differentiation

If your students need more time to complete the exercise, have them skip the bubblesearch exercise.

Have several shuffled decks of cards available for students to sort if they finish the exercise early. You can time student sorting and turn it into a competition if you need to offer extra engagement/motivation.

Encourage students to write the algorithms for different sorting methods in their own words. If a student comes up with a particularly well-worded example, encourage them to turn it into a poster or handout for the entire class to use.

If you have enough time, and your students are following along well, have your students practice implementing sorting code. Use AP questions (like \#17 in the 2009 exam) to give students practice at recognizing a right-to-left selection sort and choosing the correct inner for loop.

1.6.6 Optional Lesson

An **alternative lesson plan** is outlined here. This lesson examines the sorting methods in greater detail, and takes at least 2 class periods to conduct. If you are pressed for time, you should not use these plans since they take more time to cover, and address the material in a more challenging way (though the activities are cleverly disguised as fun)!

1. Introduce these activities in lieu of the ones mentioned in the lesson plan above.

- Bring 8 students to front of classroom, give each a piece of paper with a unique simple number on it, which they hold so class can see.
- Pick “SortMasters” from the remainder of the class to give the 8 students instructions about how to get into smallest-to-biggest order.
- Allow the first two SortMasters to give instructions without commentary, unless it looks like they are explicitly using selection sort (move lowest/biggest number to one end) or insertion sort (move a number to correct position) or bubble sort (A and B are in wrong order, so swap), in which case name what they are doing and encourage them to be explicit in describing their instructions.
- In this first activity, do not worry about explicit comparisons of two numbers at a time, SortMasters can pick smallest/largest directly.
- If the first two SortMasters have not shown insertion or selection sort, tell the class that they’re going to look at this more systematically/algorithimically now.
 - If selection sort not covered– ask a new SortMaster who should go first? Who should go second (selection sort)?
 - If insertion sort not covered – ask a new SortMaster to start with asking whether second person should go before or after first? Where third person should go relative to first and second?
- Students back to seats; ask for a comparison of selection vs. insertion sort:
 - Selection: Who goes in the next position?
 - Insertion: Into which position does the next person go?

2. **Activity 1A:** Introduce idea that while SortMasters could eyeball everyone to see who goes next, computers can only compare two things at a time.

- Distribute balances and weights. Instruct each team to start with weights 1-8 in order. Activity 1A is essentially same as Activity 1, except you explicitly tell students to use the idea of Selection Sort when ordering their weights.
- Students discuss questions in their groups, answering questions including number of comparisons.

3. **Activity 1B:** Students then move on to Activity 1B, same as 1A, except examining insertion sort. Students should be explicitly told to reset their weights in initial order.

- As a trick, set up initial weights for most groups in random order, but at least one group should have initial positions in ascending weight order, and another with weights in descending weight order.
- Come back to class discussion. Expect groups to have similar numbers of compares for selection sort, but widely different numbers for insertion sort.
- As a “grand reveal,” announce the different start conditions, so students will discover this significant behavior difference between selection and insertion.

1.7 Teacher Prior CS Knowledge

- Many students learn bubble sort as one of the first sorting algorithms. Given bubble sorts large amount of swapping in the worst case, it is a terrible sorting algorithm. However, if you ask students to form a

line creating a random list and then to get in size place order, the algorithm they usually use is some sort of bubble sort where each student is compared to the student next to them.

- When given a deck of random cards and asked to sort them, students usually come up with either selection sort or insertion sort. However, the selection sort used to sort the random cards is different from the actual algorithm used. Selection sort swaps the next smallest (or largest) item in the list. However, when students select the next smallest item, they insert it in the cards already sorted and shift the remaining cards down. You can implement a selection sort in this manner, it takes a much larger number of swaps.

1.8 Misconceptions

Selection sort requires the use of indexes in an array in order to keep track of the next lowest (or highest) item. This requires students to think in one more level of abstraction. Often times students will start with saving the number and then realize they need the index. What they end up with is two variables, one for the saved number and one for the index.

1.9 Video

- BJP 13-2, *Sorting*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c13-2
- CSE 143, *Selection Sort* (4:43:14–8:50)
<https://www.youtube.com/watch?v=CTDtbbCKmSY&start=283>
- CSE 143, *Insertion Sort* (9:26–12:00)
<https://www.youtube.com/watch?v=CTDtbbCKmSY&start=566>
- CSE 143, *Bubble Sort* (optional) (14:04–16:18)
<https://www.youtube.com/watch?v=CTDtbbCKmSY&start=844>

1.10 Forum discussion

Lesson 7.02 Sorting Algorithms (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 7.03 — Elevens Lab

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Complete** a long-form lab, demonstrating effective use of object-oriented program design, program implementation and analysis, and standard data structures and algorithms.

1.1.2 Assessments — *Students will...*

- **Complete** the Elevens Lab

1.1.3 Homework — *Students will...*

- A list of homework assignments is provided below.

1.2 Materials & Prep

- **Elevens Lab Teacher's Guide**
- **Classroom copies** of the *Elevens Lab Student Guide*
- **Associated Elevens Lab Activity Starter Code**

Read through the Teacher, Student, and Extension guides ahead of time to familiarize yourself with the parts of this long-form lab. Using the guides, complete the lab on your own to spot possible challenges for your students. Since later starter code packages include answers to the earlier sections of the lab, we recommend that you **do not** upload all student files onto computer desktops for student access. If possible, email *ActivityN Starter Code* to students the day of the lab. Otherwise, upload files manually to each desktop before each class period.

1.3 Pacing Guide: Day 1

| Section | Total Time |
|---------------------------|------------|
| Student Activity 1 | Full class |

| Homework:

- Read BJP 13.1 "Shuffling." (**Check Differentiation for advanced homework assignment and alternate classroom activities**)

Complete self-check questions \#16–21, 23–24 | TONIGHT

1.4 Pacing Guide: Day 2

| Section | Total Time |
|---|------------|
| Student Activity 2 | Full class |
| Homework:
<i>Read section 13.3 skip "Recursive Binary Search"</i>
<i>Complete self-check questions \#27–30</i> | TONIGHT |

1.5 Pacing Guide: Day 3

| Section | Total Time |
|--|------------|
| Student Activity 2, continued | Full class |
| Homework:
<i>Summarize notes and fill in missing days</i>
<i>for notebook check tomorrow.</i> | TONIGHT |

1.6 Pacing Guide: Day 4

| Section | Total Time |
|--|--------------|
| Student Activity 3
<i>See notes for leading classroom discussion below</i> | Full class |
| Notebook Checks | During class |
| Homework:
<i>Outline Chapter 13</i> | TONIGHT |

1.7 Pacing Guide: Day 5

| Section | Total Time |
|---|-------------------|
| Student Activity 3, continued | Full class |
| Notebook Checks | During class |
| Homework:
<i>Read and highlight Chapter 8 (8th or later: ch 9) of Barron's (OPTIONAL)</i> | TONIGHT |

1.8 Pacing Guide: Day 6

| Section | Total Time |
|---|-------------------|
| Student Activity 4 | Full class |
| Grade student outlines | During class |
| Homework:
<i>Take the Chapter 8 (8th or later: ch 9) exam in Barron's review book.</i>
<i>Grade your answers. (OPTIONAL)</i> | TONIGHT |

1.9 Pacing Guide: Day 7

| Section | Total Time |
|---|-------------------|
| Student Activity 5 (OPTIONAL) | Full class |
| Check Barron's Review books for highlighting, note taking, and practice test completion and correction. (OPTIONAL) | During class |

1.10 Pacing Guide: Day 8

| Section | Total Time |
|---|-------------------|
| Student Activity 5, continued (OPTIONAL) | Full class |
| Check Barron's Review books for highlighting, note taking, and practice test completion and correction. (OPTIONAL) | During class |

1.11 Pacing Guide: Day 9

| Section | Total Time |
|--|-------------------|
| Student Activity 6 | Full class |
| **Check Barron's Review books for highlighting, note taking, and practice test completion and correction. (OPTIONAL) | During class |
| Homework:
<i>Correct all homework & classwork assignments for resubmission and grading</i> | TONIGHT |

1.12 Pacing Guide: Day 10

| Section | Total Time |
|--|-------------------|
| Student Activity 7 | Full class |
| Homework:
<i>Correct all homework & classwork assignments for resubmission and grading</i> | TONIGHT |

1.13 Pacing Guide: Day 11

| Section | Total Time |
|---------------------------------------|-------------------|
| Student Activity 8 | Full class |
| Re-grade corrected assignments | During class |

1.14 Pacing Guide: Day 12

| Section | Total Time |
|---------------------------------------|-------------------|
| Student Activity 9 | Full class |
| Re-grade corrected assignments | During class |

1.15 Pacing Guide: Day 13

| Section | Total Time |
|---|-------------------|
| Student Activity 9, continued | Full class |
| Re-grade corrected assignments | During class |
| Homework:
<i>Submit 5 questions via electronic survey for test review</i> | TONIGHT |

1.16 Pacing Guide: Day 14

| Section | Total Time |
|---------------------------------------|-------------------|
| Student Activity 10 (OPTIONAL) | Full class |
| Re-grade corrected assignments | During class |

1.17 Pacing Guide: Day 15

| Section | Total Time |
|---------------------------------------|-------------------|
| Student Activity 11 (OPTIONAL) | Full class |

1.18 Pacing Guide: Day 16

| Section | Total Time |
|---------------------------------------|-------------------|
| Student Activity 11 (OPTIONAL) | Full class |

1.19 Procedure

All guides, sample code, answer code, and example code may be found in the folder “Milestone 3 Elevens Lab.”

1. To help students start the lab smoothly, start Activity 1 as a whole group.
2. Encourage students to use their Tricky Code Cheat Sheets, 4 Commandments of Scope, notebooks, textbooks, classroom posters, and homework assignments.
3. Offer occasional time-checks to help keep students on pace.
4. Grade notebooks and review books in between helping students so students can keep notebooks for homework and studying in the evenings.

1.19.1 About Barron’s

Barron’s is an AP CS A review book that some schools provide students. If your school doesn’t provide Barron’s there are many alternative homework assignments that can be found at codingbat.com/java.

Alternatively, you can save time spent on the lab by checking activities as homework.

1.19.2 Notes for Introduction Lecture for Activity 3

1. The teacher’s guide recommends leading the activity with a discussion on what makes a good shuffling algorithm.

- The Collections class has a method called shuffle that accepts a list as its parameter, and rearranges its elements randomly:

```
Collections.shuffle(list); // Where list is the name of the array you want to shuffle.
```

//

- Ask students what System.out.println method they could call to get the top card (or first element) of the array.

```
System.out.println("Top card = " + list.get(0));
```

//

1.20 Accommodation and Differentiation

Each day that you begin the lab, start with a quick survey of student concerns and questions. As needed, allow students to pair up to help each other with reading comprehension (but remind students that they each must submit their own code).

- Adaptations for group work can be found on page 17 of the Teacher’s guide.

In ELL classrooms, read all directions aloud before breaking into individual practice, and allow up to twice the amount of time for completion of the lab.

- To save time on the lab, skip lessons marked as optional.

Encourage advanced students to work through the optional lab activities. Otherwise, these students can serve as student TAs, helping others when they get stuck on code. Remind student TAs not to give answers directly, but to ask leading questions and modeling solutions to similar problems.

1.21 Forum discussion

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 7.04 — Review

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Identify** weaknesses in their Unit 7 knowledge.

1.1.2 Assessments — *Students will...*

- **Create** a personalized list of review topics to guide tonight's study session.

1.1.3 Homework — *Students will...*

- **Study** for tomorrow's test using targeted review list

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and marker**
- **Results** from electronic survey of review topics

Once students have submitted their review requests, assemble those topics into categories and prepare to re-teach the topics as needed.

1.3 Pacing Guide

| Section | Total Time |
|-----------------------------|------------|
| Bell-work and attendance | 5min |
| Review of student questions | 40min |
| Check student study lists | 5min |

1.4 Procedure

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Review of Student Questions [30 minutes]

1. Clearly indicate that you expect all students to have a list of review topics to study this evening. Periodically remind students that this list will be checked at the end of class.
2. Begin with a review of student-submitted questions before reviewing the practice questions.
3. Finally, work through the various review topics, prioritizing questions that popped up the most.
 - Some questions you may address while working through the sample test.
 - Be ready for additional questions to pop up as you go. Save yourself the work and use old homework questions and student-generated test questions as examples to work through.
4. Use a combination of group-solving questions on the whiteboard, think-pair-share, and timed-response as review strategies.
5. After you've completed reviewing an idea, remind the class that they should write down that topic if they feel they still have to review it tonight.

1.4.3 Check Student Study Lists [5 minutes]

Spend the last 5 minutes of class checking each student's review topic list as a class participation grade.

1.5 Forum discussion

Lesson 7.04 Unit 7 Review (TEALS Discourse account required)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 8.00 — Test Review & Reteach

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Re-learn or strengthen** content knowledge and skills from Unit 7.

1.1.2 Assessments — *Students will...*

- **Re-submit** test answers with updated corrections for partial or full credit
 - Credit depends on instructor preference

1.1.3 Homework — *Students will...*

- **Read** BJP 12.1 up to “Structure of Recursive Solutions”
- **Correct** any incorrect test answers by re-answering on a separate sheet of paper
 - To get back credit, they must justify their new answers
 - Staple new answer sheet to old test and turn in tomorrow

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Corrected student tests**
- **Student grades** (posted online, emailed to students, or handed back on paper in class)
- **Digital copy of test questions** for projector

1.3 Pacing Guide

| Section | Total Time |
|--------------------------------------|------------|
| Bell-work and attendance | 5min |
| Class discussion (if needed) | 10min |
| Test review and reteach | 35min |
| Check student notes and return tests | 5min |

1.4 Procedure

Return student grades before class begins or while students are completing the bellwork.

Do not return students' tests before the review session, since you want to motivate students to pay attention to the entire review, taking supplemental notes the entire time.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Class Discussion (if needed) [10 minutes]

1. If grades are low, invite the class to a discussion of what can be improved. With your co-teachers and/or TAs you should decide how to shift focus as the AP test is right around the corner. With your students, you should follow the same post-mortem format as in other review units, but with the AP exam in mind.
 - Do your students want to focus on Section II test taking strategies?
 - Perhaps they feel they need to drill quick-response Section I questions?
 - As a sanity-check, students should be reminded that they only have 1.5 minutes to solve each Section I question on the AP. If they are note near this pace, or if this is an unrealistic goal (due to language and/or reading barriers), decide as a class to focus on test-taking strategies (skipping, guessing, process of elimination) to reduce anxiety and recoup some potentially lost points.
2. Once you feel that a dialogue has been established, validate students' feelings then challenge them (e.g. AP courses are stressful, but this is good practice for college, where the pace is faster and professors don't give personalized instruction). Students can get very discouraged during this time of year. Inspire and amuse your class by pointing out old word walls or assignments (if you still have them up), showing students how far they have come since the beginning of the school year.

1.4.3 Test Review and Reteach [30 minutes]

1. Walk the students through each question on the test, glossing over questions that everyone answered correctly.
 - You can ask for students to volunteer answers, or call on students randomly. Make sure that students explain their logic when they answer. If a student gives an incorrect answer, the explanation will tell you what you need to re-teach or clarify.
 - Do not skip questions that everyone answered correctly, but do not spend more than the time it takes to read the question, and congratulate students' correct answers.
2. Project a copy of each question as you review—this will help students recall the question/process the information.

3. Make sure that students are taking notes during the re-teach, reminding students that for homework, they will have an opportunity to win back some of the points on their exam.
4. For Section II questions, select a sample of student work (with any identifying information obscured), and work through the answer together as a class.

1.4.4 Check student notes and return tests [5 minutes]

At the end of class, check student notes, and return the tests in hard copy form if applicable.

1.5 Accommodation and Differentiation

Encourage advanced students to take on additional programming challenges. One easy way to do this is to assign Programming Projects from the blue pages at the end of each Chapter.

If you have a few students that are struggling with the class, choose these students to create your classroom posters after school or for extra credit.

1.6 Forum discussion

Lesson 8.00 Test Review & Reteach (TEALS Discourse account required)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 8.02 – Writing Recursive Solutions

1.1 Overview

1.1.1 Objectives – *Students will be able to...*

- **Identify** recursive methods.
- **Predict** the output (or return value) of recursive methods.

1.1.2 Assessments – *Students will...*

- **Evaluate** statements and **predict** output during a game of grudgeball

1.1.3 Homework – *Students will...*

- **Read** BJP 12.2
- **Complete** self-check questions #5, 7-9 and exercise #1

1.2 Materials & Prep

- **Projector and computer** (optional)
- **White paper and markers**
- **Rules** for grudgeball (see website for details: <http://toengagethemall.blogspot.com/2013/02/grudgeball-review-game-where-kids-attack.html>)
- **Team assignments** that divide your class into 5 or 6 teams
- **Nerf hoop & ball** (or wastepaper and trash can)
- **Taped 2- and 3-point lines**

Briefly review the rules of Grudgeball if you have forgotten them. If you have removed your 2 and 3 point lines from last time you played, test out your 2 and 3 point lines before class begins. If you do not wish to play Grudgeball in your classroom (or if you are unable to), most Grudgeball questions can be found in the self-check question bank for 12.1.

1.3 Pacing Guide

| Section | Total Time |
|------------------------------|------------|
| Bell-work and attendance | 5min |
| Introduction and note-taking | 15min |
| Activity: Grudgeball | 35min |

1.4 Procedure

To hook your class for today's material, and if space and whiteboard setup allow, set up the grudgeball "court" and scoreboard before class begins. Remind students that lecture content will be tested during the game.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction and note-taking [10 minutes]

1. Begin by asking students what makes a method recursive (it calls itself). Ask students to offer suggestions as to how to write a recursive method. Using pseudocode (or actual code, if they offer it), outline their suggestion on the board. It should look something like this:

```
public static void writeStars (int x) {
    writeStars(x - 1);      // Prints a star.
}
```

2. Don't worry if students don't include a base case! Congratulate your students on remembering that the method calls itself, then ask students how this method is supposed to stop. (It won't! This is called infinite recursion.) To make sure that you write recursive methods that work, you need to remember 2 key ingredients:

- **Base case:** a case within a recursive solution that is so simple, it can be solved without needing to call the method again (a recursive call).
- **Recursive case:** A case within a recursive solution that involves reducing the overall problem to a simpler problem of the same kind that can be solved by a recursive call.

3. Add these 2 ingredients to the example you currently have on the board. It should look something like this:

```
public static void writeStars (int x) {
    if (x == 0) {
        // This is the base case: "write 0 stars" needs no additional method.
        System.out.println();
    } else {
        // This is the recursive case: write one star, then write however many
        // stars are left.
        System.out.println("*");
        writeStars(x - 1);
    }
}
```

4. Emphasize to students that you can write more than one recursive case, but you must always have at least 1 base case and 1 recursive case, or the code won't work. (Because you need both types of cases, recursive solutions are often written as if/else or nested statements.)
5. Ask students to explain what doesn't work about this code, and ask them to correct the recursive code here:

```
public static void writeStars (int x) {  
    System.out.print("*");  
    writeStars(n-1);  
}
```



There's a recursive case, which is good, but there is no base case! This causes infinite recursion since it has no way of stopping. Instead of stopping at 0 stars (which is what the base case would be), the code will try to write -1, -2, -3... stars forever!

1.4.3 Activity: Grudgeball [35 minutes]

1. Divide students into their assigned teams.
2. Review the rules for grudgeball, and have the students repeat the rules back to you.
3. Using the problems listed below (and any you may add, depending on your class' needs), play grudgeball until a team wins, or until the class period ends.
 - o If a class gets the answer wrong, BRIEFLY pause the game to have students offer corrections before moving to the next team's question.
 - o If correction seems to be dragging on, jump in and quickly re-teach using the incorrect answer as your example. It is important to keep the pace going to maintain student interest in the game!
4. Grudgeball problems & answers have been grouped assuming that you have 6 teams. If you have fewer teams, each "round" will be shifted accordingly, so you may have rounds where different teams are practicing different concepts. Judge each team's knowledge gaps, and adjust which questions you ask each group accordingly.
5. Questions for your Grudgeball game are listed below:

1.4.4 GRUDGEBALL PROBLEMS

1.4.4.1 Conceptual Questions

- a) What is recursion?
- b) How does a recursive method differ from an iterative method?
- c) What do you look for in a recursive method? (What parts does it have?)
- d) What is a base case?
- e) What is a recursive case?
- f) Does a recursive method need to have both base and recursive cases?
- g) Why does a recursive method need to have a base case and a recursive case?
- h) Can a recursive method have more than one base case?
- i) Can a recursive method have more than one recursive case?
- j) What happens if a recursive solution is missing a base case?

1.4.4.2 Predict-the-Output Questions

Use the following method for questions k – s:

```
public static void mystery1 (int n) {  
    if (n <= 1) {  
        System.out.print(n);  
    } else {  
        mystery1(n / 2);  
        System.out.print(", " + n);  
    }  
}
```



What is the output produced by the method call indicated?

- k) mystery1(1);
- l) mystery1(2);
- m) mystery1(3);
- n) mystery1(4);
- o) mystery1(16);
- p) mystery1(30);
- q) mystery1(100);
- r) mystery1(-1);
- s) mystery1(2.2);

1.4.4.3 Predict-the-Output Questions (continued)

Use the following method for questions t – x:

```
public static void mystery2 (int n) {  
    if (n > 100) {  
        System.out.print(n);  
    } else {  
        mystery2(2*n);  
        System.out.print(", " + n);  
    }  
}
```

//

What output is produced by the method call indicated?

- t) mystery2(113);
- u) mystery2(70);
- v) mystery2(42);
- w) mystery2(30);
- x) mystery2(10);

1.5 Accommodation and Differentiation

In ELL classrooms, read the questions aloud in addition to showing the question on the board or projector. Consider distributing a worksheet with the questions on it so students can write down answers during the game.

If students are having difficulty with the “predict the output” questions, a step-by-step explanation of how a recursive method executes can be found here (<http://tinyurl.com/lablr3h>). You can pattern your explanations/re-teaching to students using the same method.

1.6 Misconceptions

- When the recursive call is made, mistakenly thinking the same parameter is being used instead of a distinct copy for the recursive call.
- The only time the base case is called is if the initial call is the base case.

1.7 Video

- BJP 12-2, *Implementing a Recursive Function*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c12-2

1.8 Forum discussion

Lesson 8.02 Writing Recursive Solutions (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 8.03 — Mechanics of Recursion

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Model** how recursive methods execute.

1.1.2 Assessments — *Students will...*

- **Write** a recursive method
- **Model** the execution of that method for the instructor
- **Model** a method written by their peers

1.1.3 Homework — *Students will...*

- **Complete** self-check \#6, 10 and exercise \#3

1.2 Materials & Prep

- **Projector and computer** for hook (https://youtu.be/3wbvs_n2oty)
- **Whiteboard and markers**
- **Classroom copies** of [WS 8.3, Teacher Demo 8.3](#)
- **One pair of scissors** for each group
- **Small group assignments** (~4 students per group)
- **Classroom copies** of the textbook (**optional**)
- **Printout** for Teacher Demo 8.3 (**optional**)

Carefully read through the “Mechanics of Recursion” example in section 12.2. You should make a copy of WS 8.3 for yourself and do today’s activity before delivering the lesson so you can smoothly demonstrate (and check) how to model a recursive method. If you decide to model an example for the class before the activity, use the pre-printed example so students can’t just copy their sample method from the book.

1.3 Pacing Guide [Optional, Lessons Vary]

| Section | Total Time |
|--|------------|
| Bell-work and attendance | 5min |
| Discussion of hook & introduction | 15min |
| Activity 1: Modeling Recursive Methods | 20min |
| Activity 2: Modeling a Peer's Methods | 10min |

1.4 Procedure

On your classroom projector or screen, have the animation of the Dragon Curve playing on a loop (link in "Materials and Prep"). Adjust the player to play 2x speed to keep student interest; you may want to mute the music. As bell-work, ask students how this animation is an example of recursion. Their explanation should describe the recursive case in pseudocode.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Discussion of Hook & Introduction [15 minutes]

1. Begin by reviewing the hook/Bellwork with students. There may be some debate as to what the recursive and base cases are for this curve; allow time for students to argue and come to consensus. Students should understand that (A) the base case is just a single straight line, and (B) the recursive case involves repeated the line with a 90° turn.
2. There's an important distinction to be made here: the animations linked to from this lesson show recursion from the "bottom up." This is not the way that recursion works. The modeling activity is a better representation of the recursive process; the animations are meant only to demonstrate how even the simplest recursive method can quickly yield large, complex results.

If you feel it might be helpful to show your students a recursive implementation of drawing the dragon curve, you can find one here: <https://scratch.mit.edu/projects/65378560>.

If your class is fairly advanced, you may want to skip Teacher Demo 8.3. Advanced students will benefit from exploring the model on their own as outlined in WS 8.3. However, for many classes additional scaffolding will be needed.

3. To perform the demo, you should:

- Cut out the cards in the Teacher Demo 8.3
- If you have a classroom projector, do the demonstration under the capture so your cards are magnified to the point where students in the back of the classroom can see what you are doing. If you do not have a magnifying projector, you may want to do this demo as you walk around the class, showing students what is on each card.
- Explain to students that you are modeling Java's call stack. A call stack is the way that Java keeps track of the sequence of methods that have been called. The model your using today is another way of tracking flow-of-control.
- Tell students that in your example you'll be modeling a method that reverses a list of words. When students break into groups for the activity, they will be writing their own recursive methods and modeling Java's call stack for the methods that have been called.
- Briefly review the reverse method introduced in 12.2.

4. Show students the first piece of paper from Teacher Demo 8.3 with the method definition.
 1. This should look pretty familiar to students, but have them point out to you the recursive case (`input.nextLine()`) to review.
 2. Point out to students the new feature on your model that holds the local variable line. Ask students what they think this represents? (It is a place in the computer's memory that stores the variable line.)
 3. Ask students to narrate what happens when we call the reverse method.
 4. Write the `this` variable on the slip of paper, and ask students what happens next. The method will get called again—but because recursion works differently than iterative code, we don't just run through the method again like with a loop.
 5. Take out the second copy of the reverse method, and place it over the first paper, as indicated in the textbook. Have students walk you through the method again and tell you what local variable will get written in the storage space at the bottom of the paper ("is").
 6. Repeat these steps until you have reached the base case (the fifth sheet of paper). Ask students what the local variable is, and discuss what happens when there is no word left to reverse. (The `input.hasNextLine()` returns false, so that method terminates.)
 7. Now remove that fifth card from the pile, revealing the fourth copy of the method again. Let students know that Java returns to where it was before it executed the call that just terminated.
 1. Ask students what happens next. (We print the line "no?" and terminate.) Write this output on the board so students can keep track of the output.
 2. This means that Java now goes to the place where it was before it terminated this call. Ask students what you should do now in your modeling of this recursive method. (Take off this card, and reveal the third method card.)
 8. Continue in this way until you have returned to the first card.
 9. If you need to repeat sections of this demo, do so until you feel that 50% of your class understands the process. At this point, students should be able to help each other learn the concepts during their activity.

1.4.3 Activity 1: Modeling Recursive Methods [20 minutes]

1.4.3.1 Emphasize with students

1.4.3.2 Content - Advanced programming structures - Managing Complexity

Recursion is an interesting but sometimes difficult concept. When dealing with more advanced ideas like this, it's important to manage complexity in order to design and implement complex programs.

This activity helps you organize recursion in your program. It's advised that you use these ideas, and consider these questions, as you create more programs that implement recursion.

-
1. Direct students to read through all the directions first before beginning the activity. Point out to them that they will be completing their own demo (similar to the one you just completed) using their own code.
 2. Break students into small groups and distribute worksheets and scissors.
 3. Keep students on pace by announcing 10- and 5-minute warnings.

4. If students complain about having to rewrite their recursive code so many times, point out that it's a good thing that we only have to type out the recursive code once! In reality, the method doesn't exist as many copies (that would take up a lot of memory!) We're just using physical examples to keep track of how Java is calling each "round" of the recursive method.
5. As students work, check first for correct recursive methods, then make a second round having students perform their modeling activity for you.
6. Be sure each group explains:
 - Where the base case is, and when the base case occurs.
 - Where the recursive case is.
 - What causes the method to advance to the next round of recursion.
 - What causes the method to terminate.
 - Where Java returns to after a method has terminated.
 - What the output is.
7. If students do not answer all of these questions correctly, give them time to regroup and assess them again after they have corrected their thinking.

1.4.4 Activity 2: Modeling a Peer's Methods [10 minutes]

1. After 20 minutes (or when students finish), have groups trade code sheets, and challenge the groups to work their way through the code again.
2. Keep them on pace by announcing time (they only have 10 minutes for this second activity). Visit each group and ask for groups to model these new methods for you.
3. You can opt to select some student groups that have a smooth presentation to share with the entire class before class dismisses.

1.4.5 Activity 3: Iterative to Recursive and Vice-Versa [Optional]

1. Introduce iterative and recursive methods that produce the same result. For example, here is an iterative and recursive example for solving factorials:

| | |
|--|---|
| <pre>// Recursive int factorial(int n) { if (n == 1) { return 1; } else { return n*factorial(n-1); } }</pre> | <pre>// Iteration int factorial(int n) { int product = 1; for (int i = 2; i <= n; i++) { product *= i; } return product; }</pre> |
|--|---|

2. Invite your class to take a recursive function and turn it into an iterative function. You can provide a new recursive function or they can use one's they've previously worked with. To help your class, put the recursive step into a for loop that worked to reach the base case [*i* (=>) (\#)].
3. Discuss what needed to change between the recursive and iterative calls.
4. Now, ask your class to do the inverse; take an iterative function and turn it into a recursive one. Based off your discussion and the previous change between recursion and iteration, this task should be easier.

1.4.6 Activity 4: Creating Fractals with Recursion [Optional]

1. Explain to your students that the dragon curve we worked with earlier is called a fractal, a fractal being a geometric figure in which each part has the same statistical character as the whole. They follow a pattern in which similar patterns recur at progressively smaller scales.
2. Invite your class to create the dragon curve in pseudocode, they should notice that the figure (they don't know how to draw) rotates 90° around its near point.
3. Now, as an easier activity, we're going to play with the Koch curve, which the code can be found here on stack: <http://stackoverflow.com/questions/13000994/koch-snowflake-java-recursion>. Explain all parts of this program, specifically the level, turning, the recursion in drawKochCurve, and the gpdraw package being used (myPencil and myPaper).

The Koch curve follows three rules:

1. Divide the line segment into three segments of equal length.
 2. Draw an equilateral triangle that has the middle segment from step 1 as its base and points outward.
 3. Remove the line segment that is the base of the triangle from step 2.
4. As a fun lesson, let your class play with the values, altering the Koch curve to create new designs. Award prizes to special and unique fractals students create or require that a fractal (and its code) be turned in at the end of class. If your class is comfortable with the activity, you can introduce color for myPencil which opens a door for interesting designs.

1.5 Accommodation and Differentiation

While all students should write their OWN algorithm, you should encourage students to work in pairs or small groups so they can share ideas and help each other organize their thoughts. This is particularly important in ELL classrooms, where emergent English speakers can pair with advanced English learners. If some students want to do this project all on their own, let them.

If you have students who are speeding through this lesson, you should encourage them to:

- Create a mnemonic or acrostic to remember all the steps for checking syntax errors
- Make a poster for the classroom illustrating the mnemonic or acrostic
- Help another student with the worksheet (explain, not solve-for-them)

1.6 About Error Checking in

If your students are enthusiastic about the Dragon Curve, expand on the discussion by pointing out that there are 2 repetitions in the Dragon curve, one left and one right. This means that there are 2 lists in the recursive code.

Ask students to speculate on what the curve would look like with 4 repetitions. An animation of a 3D dragon curve can be found here: (<https://youtu.be/bnutikyricu>)

For less-advanced classrooms, you may want to pre-populate the group worksheets with recursive methods. This will allow students to focus on processing the execution of the method rather than struggling with syntax. This is not recommended unless you are very pressed for time. In most cases, students should be practicing properly writing recursive code.

1.7 Video

- BJP 12-3, *Implementing a Recursive Method*
http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoplayer.php?id=c12-3

1.8 Forum discussion

Lesson 8.03 Mechanics of Recursion (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 8.04 — MergeSort

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Use** mergeSort to sort an ArrayList.
- **Use** recursion to traverse and Array.

1.1.2 Assessments — *Students will...*

- **Build** a merge algorithm to be applied in mergeSort.

1.1.3 Homework — *Students will...*

- **Summarize** notes for notebook check tomorrow

1.2 Materials & Prep

- **Projector and Computer**
- **Whiteboard and Markers**
- **Electronic survey** for student review requests

The homework tonight asks students to submit 2 questions for review; the number is reduced for this until since there is far less material than usual. Create an electronic survey for students to complete with 3 text fields, one for name, and 2 for questions they have about Ch. 12 content. Set a deadline by which time students must have submitted 2 questions from Ch. 12 If students do not have questions, stipulate that they still have to submit something to receive credit, even if it is only questions they think other students may have.

1.3 Pacing Guide

| Section | Total Time |
|--|------------|
| Bell-work and attendance | 5min |
| Introduction and homework distribution | 5min |
| Student work | 35min |
| Students trade work, check, and submit | 10min |

1.4 Procedure

Today we will return to sorting and cover MergeSort. MergeSort acts recursively; it takes a list, splits it down into single elements, and compares the elements, merging them together in an orderly fashion. The first part of the lesson covers merge as it's conceptually easier, the second part of the lesson covers the splitting of the list.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Introduction to MergeSort [5 minutes]

1. Begin your lecture by bringing the class back to selection and insertion sort. Make the case that selection and insertion sort both work, but take more time for your computer to process (graph in slides).
2. Show your class an example diagram of mergeSort; they should realize (with your help) that the worst case performance for mergeSort is faster than either selection or insertion sort.

1.4.3 Merge [15 minutes]

1. Invite your class to try and merge two sorted lists of numbers. How would we compute a single sorted list containing all the numbers in list 1 and list 2? Have your class set up the pseudocode.
 - We must maintain an index for each list starting at 0.
 - We must create an empty list to hold the result.
 - When we haven't exhausted our two lists, insert the smallest element at the point in the new list and advance the index.
2. Let your class work for 10 minutes to try and create the merge function. Using the responses they come up with, lead them to the correct merge function. (Shown on slides, using `ArrayList<integer>`)

1.5 MergeSort [30 minutes]

1. Challenge your class to try and implement merge into a sorting algorithm. Those who have done the reading should have an idea about where to get started, but may get stuck on the recursive portion.

You can give them these instructions to get started:

1. If the list's size is 0 or 1, just return the original list (as it is already sorted)
2. Split the list parameter into two lists, of (roughly) equal size: list 1 and 2.

3. Sort both list 1 and 2.
 4. Merge the two sorted lists, and return the result.
2. For lab today, your class will be tasked with making this mergeSort algorithm. At the end of class post the solution up on the board.
- Provide hints throughout the process, namely:
1. When the list's size is 0 or 1 and you return the original, that is your base case.
 2. Split your lists based off ArrayList.size().
 3. The merge function does merging.
3. If the class is struggling, walk through the entire mergeSort algorithm with them, mergeSort is covered by the AP so a base level of understanding is important.

1.6 Accommodation and Differentiation

In ELL classrooms, pair students and allow them to work together to correct their work. If you noticed a particular problem was difficult for the majority of students, read the question aloud and help students work through it.

For those students who have nothing to correct (or finish very early), reward them with silent free time, or allow them to work on a free-choice programming project.

1.7 Video

- CSE 143, *Merge Sort* (12:09–14:04)
<https://www.youtube.com/watch?v=CTDtbbCKmSY&start=729>
- CSE 143, *Merge Sort* (20:50–26:15)
<https://www.youtube.com/watch?v=CTDtbbCKmSY&start=1250>

1.8 Forum discussion

[Lesson 8.04 MergeSort \(TEALS Discourse account required\)](#)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 8.06 — Review

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Identify** weaknesses in their Unit 8 knowledge.

1.1.2 Assessments — *Students will...*

- **Complete** Quiz 8.5.

1.1.3 Homework — *Students will...*

- **Read and highlight** Study for Quiz

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and marker**
- **Results** from electronic survey of review topics

Once students have submitted their review requests, assemble those topics into categories and prepare to re-teach the topics as needed. Rather than using an entire class period for a test, we're condensed the Chapter 12 topics into a quiz. This should allow you more time to start reviewing for the AP exam.

1.3 Pacing Guide

| Section | Total Time |
|-----------------------------|------------|
| Bell-work and attendance | 5min |
| Review of student questions | 40min |

1.4 Procedure

At the beginning of class today:

- Remind students that the quiz is at the end of class today.
- Review student-submitted questions.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Review of Student Questions [30 minutes]

1. Work through the various review topics, prioritizing questions that popped up the most.
 - Some questions you may address while working through the sample test.
 - Be ready for additional questions to pop up as you go. Save yourself the work and use old homework questions and student-generated test questions as examples to work through.
2. Use a combination of group-solving questions on the whiteboard, think-pair-share, and timed-response as review strategies.
3. After you've completed reviewing an idea, remind the class that they should write down that topic if they feel they still have to review it tonight.

1.4.3 Check Student Study Lists [5 minutes]

Spend the last 5 minutes of class checking each student's review topic list.

1.5 Accommodation & Differentiation

The quiz features a recursive method to produce the Fibonacci sequence. If your students need extra help/scaffolding, you might want to show students this YouTube video (<https://youtu.be/dsmbruczs7k>) and lead a class discussion to ensure students are familiar with the math involved.

1.6 Forum discussion

[Lesson 8.06 Review & Quiz \(TEALS Discourse account required\)](#)

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 8.08 – Quiz Review & Reteach

1.1 Overview

1.1.1 Objectives — *Students will be able to...*

- **Re-learn or strengthen** content knowledge and skills from Unit 8.

1.1.2 Assessments — *Students will...*

- **Re-submit** test answers with updated corrections for partial or full credit
 - Credit depends on instructor preference

1.1.3 Homework — *Students will...*

- **Complete** the assessment in Chapter 7 of Barron's Review Book
- **Correct** any incorrect quiz answers by re-answering on a separate sheet of paper
 - To get back credit, they must justify their new answers
 - Staple new answer sheet to old test and turn in tomorrow

1.2 Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Corrected student tests**
- **Student grades** (posted online, emailed to students, or handed back on paper in class)
- **Digital copy of test questions** for projector

1.3 Pacing Guide

| Section | Total Time |
|--|------------|
| Bell-work and attendance | 5min |
| Quiz review and reteach | 30min |
| Return quizzes/start homework assignment | 15min |

1.4 Procedure

Return student grades before class begins or while students are completing the bellwork.

Do not return students' quizzes before the review session, since you want to motivate students to pay attention to the entire review, taking supplemental notes the entire time.

1.4.1 Bell-work and Attendance [5 minutes]

1.4.2 Quiz Review & Reteach [30 minutes]

1. If grades are low, invite the class to a discussion of what can be improved. With your co-teachers and/or TAs you should decide how to shift focus as the AP test is right around the corner. With your students, you should follow the same post-mortem format as in other review units, but with the AP exam in mind.
 - Do your students want to focus on Section II test taking strategies?
 - Perhaps they feel they need to drill quick-response Section I questions?
 - As a sanity-check, students should be reminded that they only have 1.5 minutes to solve each Section I question on the AP. If they are note near this pace, or if this is an unrealistic goal (due to language and/or reading barriers), decide as a class to focus on test-taking strategies (skipping, guessing, process of elimination) to reduce anxiety and recoup some potentially lost points.
2. Students can get very discouraged during this time of year. Inspire and amuse your class by pointing out old word walls or assignments (if you still have them up), showing students how far they have come since the beginning of the school year.
3. Walk the students through each question on the test, glossing over questions that everyone answered correctly.
 - You can ask for students to volunteer answers, or call on students randomly. Make sure that students explain their logic when they answer. If a student gives an incorrect answer, the explanation will tell you what you need to re-teach or clarify.
 - Do not skip questions that everyone answered correctly, but do not spend more than the time it takes to read the question, and congratulate students' correct answers.
4. Project a copy of each question as you review—this will help students recall the question/process the information.
5. Make sure that students are taking notes during the re-teach, reminding students that for homework, they will have an opportunity to win back some of the points on their exam.
6. For Section II-type questions, select a sample of student work (with any identifying information obscured), and work through the answer together as a class.

1.4.3 Return Quizzes/Start Homework Assignment [15 minutes]

At the end of class, if time allows, return the quizzes and allow students some time to begin their homework assignment.

1.5 Accommodation and Differentiation

Encourage advanced students to take on additional programming challenges. One easy way to do this is to assign Programming Projects from the blue pages at the end of each Chapter. Section 12.5 has an excellent

recursive graphics lab that would be a wonderful assignment for advanced students.

If you have a few students that are struggling with the class, choose these students to create your classroom posters after school or for extra credit.

1.6 Forum discussion

Lesson 8.08 Quiz Review & Reteach (TEALS Discourse account required)

formatted by Markdeep 1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 9.00 — Reviewing for the AP Exam

1.1 Topic Review & Re-teaching

To get a sense for which topics your class needs to review, we recommend administering a Barron's practice exam as a pre-test, another as a post-test after some review, then the 2009 released AP CS A exam (the most recent fully-released exam) as a final practice.

The 2009 exam includes GridWorld questions (Free response question 2, multiple choice questions 21–25). Instructors could have students skip these now obsolete questions, or replace them with questions of similar difficulty from the 2004 exam to allow an AP score to be reasonably computed.

As you re-teach concepts, practice questions and assessments can be gleaned from the Bellevue International Mastery Tests (included as a file within the Unit 9 materials). These tests, written and graciously shared by Arthur Watson, provide an easy way to retest your students. Should you decide to use these resources, keep the following in mind:

1. The tests do not include recursion, searching, or sorting since these were the most recently covered units before whole-year review.
2. There are 3 versions of each test (A, B, and C). Versions are similar, so students can re-take each exam after working on material to earn a higher score.
3. As it gets closer to AP test time, Arthur lets his class use the *A* version test/answer keys to study for the *B* and *C* versions.

1.2 Practice Questions

Whether in-class or for homework, giving students ample opportunities to drill test questions will help them overcome test anxiety. The resources listed below provide practice material using different learning modalities:

| Website URL | Features |
|---|---|
| http://APcomputersciencetutoring.com | Practice problems & solutions broken down by topic. Old AP multiple-choice questions & answers. |
| http://tinyurl.com/obwpm39 | 175 multiple-choice questions (self-checking) |
| http://tinyurl.com/nkauhfb | Timed practice tests, flashcards, featured “question of the day.” |

1.3 Test Taking Strategies

If you have been using the exams provided with this curriculum, students should be familiar with the directions and guiding text of the AP exam. During review, you should push the test-taking strategies covered in the websites referenced below:

| Website URL | Features |
|---|---|
| http://tinyurl.com/APCSTips | Official tips from the College Board. |
| http://tinyurl.com/APCSlist | Test-taking tips and Section II tips from a teacher. |
| http://tinyurl.com/APCSquickReference | See page A1 for quick reference guide included with the exam. |

1.4 Accommodation & Differentiation

There are 3 available practice exams you can give to your students. For ELL classrooms, start by giving the first practice exam with a little extra time (~ 10 minutes) on each section, so they have time to get used to pacing and reading.

After you have re-taught topics, assigned practice timed practice at home, give the next exam at full time, and ask the students to report back to you what they felt were the most challenging aspects of the exam.

If students report that reading is slowing them down, take a class period or two to drill the word wall vocabulary. Have students practice reading and explaining questions to each other. At this point, repetition is key. Strengthening topic-specific vocabulary and recall will be more beneficial than reviewing additional topics.

Some fun vocabulary review strategies include:

- Pictionary: encourage students to write some code (or find, or circle it in a sample of code on the board or projector) to illustrate a concept or term.
- Vocabulary bingo: Print out word wall words (or challenging concepts) on bingo cards (you can make them online here: <http://osric.com/bingo-card-generator>). Rather than just calling out words, call the definition of the term or project an example of the concept in sample code, having students come up with the word they need to find on the bingo sheet.

If your students are speeding through the review sessions, add additional challenges by sticking to Section II questions. Students can access Section II questions from 2009 – 2014 on the AP CS website here: <http://tinyurl.com/m7jyec>. The most recent released Section II test questions and scoring rubric are included with Unit 9 materials. 2014's questions and scoring rubric are available on the [AP web site](#).

1.5 Forum discussion

[Lesson 9.0 Reviewing for the AP Exam \(TEALS Discourse account required\)](#)

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Lesson 9.01 — Mock AP Exam

1.1 Run a Mock AP CS A Exam

It is important for students to be familiar with the exam format before taking any AP exam.

Run an Mock AP CS A exam by logging into the College Board CS Classroom and download one of released AP Exam.

The logistics of filling out the form can be done on a separate day so it does not take up time when students are answer the questions.

Break up the exam into parts that fits into your school schedule.

Ideally the multiple choice would be completed in one day and the free response on another. You may need to break each part in half depending on your class meeting time.

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 CHARACTER CLASH!

2 FINAL PROJECT

Time to get creative and apply your programming and object oriented skills to a new context!

Your task is to create a software application that is similar to the Pokemon game. It should include a number of characters that can clash with each other, or against computer opponents, in battles. The winner of each battle will be determined by a number of things including each character's attributes, attacks and defenses.

Image of a wide variety of characters including a ghost, a shark, a bee, an alien and a robot.

You will use many of the concepts that have been introduced throughout the course to establish attributes for each character and to determine hit points and health status. You will use object oriented programming concepts to implement your program.

You will follow the Applied Design stages and begin by interviewing an end-user to determine their interests, then you will design a Pokemon-like game that they would enjoy playing. The end-user will be involved in the design and testing of the game.

Your software application should include:

- A number of characters
- A number of common attributes for each character
- A variety of power levels for each attribute of a given character
- A range of attacks that each character can perform
- Formulas to determine the effectiveness of each character's attack in given situations (this may involve some randomness)
- The ability for the user to list all data related to the characters
- The ability for the user to select and play the game as one of the characters
- The ability for the user to select a variety of attacks and defenses
- Game play that progresses the user through the game and determines whether or not the user defeats other characters to win the game, or is defeated and therefore loses.

You will need to carefully select your characters and a context in which they can do battle. This will require you to understand the interests of your end user, but it will also require you to be creative and to apply many of the concepts you have learnt throughout the course.

These characters or groups will be able to "battle" each other or they can work together to battle "computer opponents". The results of each battle will be dependent on the statistics of each of the characters or groups in the battle, as well as the attacks and defenses chosen by the user. You can also add some randomization into the battle as well.

The end-user will provide you context for the program by indicating their interests and likes/dislikes. The following list provides you with some quick ideas to help you start thinking of some possibilities:

- A program that allows various bugs to battle each other, each bug has different attributes and strengths/weaknesses
- A program that allows different basketball teams to compete against each other, each team has different strengths and weaknesses (three point shooting, defense, endurance). The game allows the user to play

- as a coach, selecting what to do in specific situations
- A program that allows different African animals to battle for "Savanah Supremacy".
 - A program that simulates a number of different cars being able to race each other on different tracks (gravel, pavement, desert, rainy, etc).
 - A program that simulates a battle between vegetables, fighting to be recognized as the healthiest thing to eat.
 - A program that uses characters that you have made up yourself, with attributes, that battle for some type of championship.
 - A program that simulates a wide range of pets (cats, dogs, birds, lizards, etc) who are defending their owners home from a burglary or defending themselves from being petnapped!

You will have to be creative as you complete the program, and you will also have to carefully design and test the game play and logic of the battles. You want to ensure that the formulas for hit points and health make sense, and that the game includes an appropriate level of difficulty.

Example... The following is just an example to illustrate the various components of the program.

Talia is a student in the course. She interviews her little sister as the end-user of her game and determines that her little sister loves aliens. Talia brainstorms some ideas and presents the following idea to her sister:

A game that includes 8 aliens, one from each of the planets in the solar system. Each alien has five attributes including oozePower, hop-ability, laserShot, protectCharms and powerPunch. These attributes are rated on a scale of 1-5, with each character possessing a total score of 17.

As an example, one of Talia's characters is called MartyMarsian. He has the following levels for each of the attributes:

- oozePower: 3
- hop-ability: 4
- laserShot: 5
- protectCharms: 2
- powerPunch: 3

Notice that the values add up to 17 and no value is above 5 or below 0. This is a design feature that Talia wanted to include in her game.

Talia meets with her little sister after the initial design stage and she loves Talia's ideas. They work together to determine how each character will do battle and decide that the user will get to select which alien they play as. They will then have to battle the seven other aliens to be crowned Champion of the Solar System. The alien will receive extra points after each victory and Talia will design formulas that make the game challenging, but winnable.

Image of the planets

Talia designs a superclass and subclasses for her aliens and creates formulas (that include some random values) to determine the amount of health lost in each battle. Talia creates user menus that allow the user to navigate the program and to allow the user to see and sort the characteristics of each alien.

Talia tests the program with friends and relatives to obtain further feedback and suggestions. She then implements a few changes and submits her finished project to her teacher!

2.1 Implementation Requirements

2.1.1 Complexity and Creativity

Your final project should be sufficiently complex and large-scale to push your limits as a programmer, but not so sophisticated that you are not able to complete it in the time allotted. The complexity in your project should come from the design and the algorithms and not from the code. (That is, you cannot meet the complexity

requirement simply by writing a lot of code. Your code must be challenging or interesting in some meaningful way.) In addition, you should not add complexity by introducing peripheral elements, such as graphics or sound effects. (Your program can certainly have these, but they will not be considered in determining the projects complexity.) In addition, one of the main goals of this project is to allow you to unleash your creativity and allow you to create something of interest to you. To achieve this, your project must show some level of creativity or personalization that makes it your own. Simply creating your own version of some existing application will not fully meet this requirement. For both the complexity and creativity requirements, you should talk to the instructors early and often to ensure your project is in line with our expectations.

2.1.2 Documentation and Style

As with all previous projects, your program must be well-written, well-documented, and readable. Writing code with good style is always a good idea, but in a project of this size and scope, following style guidelines will help you keep your thoughts organized and make it easier to keep track of your progress, pick up where you left off each day, and find and fix bugs.

Resources: Throughout this course, you have completed a number of small programs that implemented a Pokemon game. You now have the opportunity to take these skills and apply them to a new context. Be sure to review course resources so that you fully understand how this type of game can be implemented. You can alter and change some of the ideas presented in the course to facilitate the completion of your own game. The following are just a few resources that you might want to review:

- 3.03 – Pokemon and returning values
- 3.06 Hitpoints and formulas
- 3.07 Pokemon Battle Program
- 3.14 Random Numbers
- Unit 5 Object Oriented Programming
- Unit 6 Inheritance and Polymorphism

These resources will help you understand how characters and attributes are implemented in Pokemon:

- Basic Overview of the Game: <http://www.pokemon.com/us/parents-guide/>
- Advanced Individual Value & Stat Calculator: <https://legendarypkmn.github.io/javacalc.html>
- Gameplay: <https://youtu.be/DlEbXH8eUTk?t=1m26s>
- Types of Pokemon: <http://www.pokemon.com/us/pokedex/>
- Pokemon with Stats: <http://tinyurl.com/no4mzic>

STEP 1 – UNDERSTANDING CONTEXT

Conduct user-centered research to understand design opportunities and barriers.

Select an end-user for whom you will design and create this program (this can be a friend, classmate, relative, etc). Create interview questions that will allow you to understand the end-user's interests and likes/dislikes. Provide these questions to your teacher and then use them to interview your end-user.

Record all of the end-user's responses to your questions.

STEP 2 – DEFINING AND IDEATING

Choose a design opportunity and point of view, make inferences about premises and boundaries, take creative risks to identify gaps to explore, generate ideas to create a range of possibilities, prioritize ideas for prototyping and designing with users

Using the responses from your end-user interview, begin to develop a plan for your program. List:

- the types of characters included in your program
- their names
- their attributes and relative "power" for each
- their strengths and weaknesses
- any other details that might be relevant for the characters
- any formulas that will determine hit points and health values during a battle
- a logical game play format

Share these ideas with your end user. Record their comments, suggestions and feedback and note any changes that you may make as a result of this interview.

Identify any issues or problems that might arise as you begin to program (what areas might require more information or programming solutions? Where can you find this information?)

CHECKPOINT 1

SUBMIT YOUR INTERVIEW QUESTIONS, RESPONSES AND PROGRAM DESIGN PLAN TO YOUR TEACHER.

STEP 3 – PROTOTYPING AND TESTING

Construct prototypes, making changes to code as needed.

Program your game. Be sure to review course notes and activities to make sure that you effectively implement object oriented programming design for your characters. Decide how battles will take place, how each character will have a chance to attack and defend, and how points will be awarded during battles. Be sure to program your game in small steps, ensuring that each step is fully functioning before adding complexity.

When you are ready, create a short test plan and test your program. Include expected output and actual output and include details related to any fixes that needed to be made.

Have your end-user try a working version of your program. Note their suggestions, comments and feedback. Indicate any changes that you made to the program, based on the user's feedback.

CHECKPOINT 2

SUBMIT THE CURRENT VERSION OF YOUR PROGAM TO YOUR TEACHER, AS WELL AS YOUR TEST PLAN AND FEEDBACK FROM THE END-USER

STEP 4 – SHARING, TESTING AND FINAL ITERATION

Gather feedback from users over time to critically evaluate design and make changes to product design or processes Identify new design issues

Share your work with classmates, friends or family. Record any feedback, suggestions and comments and use this information to make the final iteration of your program.

CHECKPOINT 3

SUBMIT THE FINAL VERSION OF YOUR PROGAM TO YOUR TEACHER, AS WELL AS YOUR FEEDBACK FROM CLASSMATES, FRIENDS OR FAMILY.

Grading Scheme/Rubric

| | |
|--|-----------|
| Implementation | |
| Project is appropriately complex and creative | 4 points |
| Program is well-documented and shows good style | 10 points |
| Final product meets all requirements and goals laid out in specifications | 8 points |
| Program uses programming concepts effectively, including all required elements with an appropriate level of complexity | 4 points |
| Object Oriented Programming concepts are effectively applied with an appropriate level of complexity | 6 points |
| Game play is logical and effective | 3 points |
| Formulas for damage/health are logical and effective | 3 points |
| Checkpoint 1 | 2 points |
| Checkpoint 2 | 2 points |
| Checkpoint 3 | 2 points |
| TOTAL | 44 points |
| — | — |
| Applied Design Steps and Log | |
| Understanding Context components are complete and thorough | 3 points |
| Defining and ideating components are complete and thorough | 3 points |
| Prototyping and testing components complete and thorough | 2 points |

| | |
|---|--------------|
| Sharing, Testing and Final Iteration components are complete and thorough | 2
points |
| TOTAL | 10
points |
| — | — |
| TOTAL | 54
points |

formatted by [Markdeep 1.093](#) 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Culture Day Lesson A: Video/Reading

1.1 Learning Objectives

Students will be able to...

- Describe the key topics discussed in the video\reading
- Answer critical thinking questions about the video\reading
- Connect the video\reading to some aspect of their culture, society, or life

1.2 Materials\Preparation

- The reading or video for the lesson
- Critical thinking and\or analysis questions relating to the topic of the lesson

1.3 Pacing Guide

| Duration | Description |
|------------|---|
| 5 minutes | Welcome, attendance, bell work, announcements |
| 10 minutes | Introduction to topic and video\reading |
| 15 minutes | Watch video\Read material |
| 20 minutes | Class discussion or activity |
| 5 minutes | Debrief and wrap-up |

1.4 Instructor's Notes

Introduction to topic

- Provide motivation or context for the day's topics by asking questions connecting the topic to students' experiences
 - e.g. "How many play video games at least 10 hours a week? Do you get anything out of playing besides entertainment? Today we're going to watch a video about the positive benefits of playing video games."

- This can be very brief to simply set up the video\reading, or it can be a deeper conversation to encourage students to consider their own positions on the subject. If taking the latter approach, some time will likely need to be moved from the “class discussion” section to this section
- Describe the topic of the video\reading and provide context on the presenter\author and the background on the specific video\reading.
 - Avoid getting into too much detail here, and *especially* avoid summarizing the material. Your goal is to provide context and background, not to preview the material itself.
- Present the critical and\or analytical questions that will guide the class discussion or activity later. Provide any necessary explanation or elaboration to ensure students understand the expectations.
 - In some cases, you will want to elaborate on the questions to get students all on the same page for the discussion or activity later. In other cases, you will want to be deliberately vague to encourage students to form their own ideas or interpretations.

1. Video/Reading

- Students watch the video\read the reading
 - Remind students to consider the guiding questions when watching\reading
 - Students may take notes if desired, but should give their full attention to the material
 - The room should be mostly silent for this part of the lesson, and students should not have access to computers
 - Some students may want to take notes on the computer, but doing so presents temptation to do other things. Paper notetaking is recommended.

1. Discussion\Activity

- Lead a class discussion or activity about the topics covered in the video\reading and guiding by the questions presented before the material. This can take one of several forms, including, but not limited to:
 - an open, full-class discussion
 - when using the approach, be sure that all students have a chance to contribute and that the conversation is not dominated by a few voices
 - small group discussions, after which one member of each group shares with the class
 - this can either be open-ended, allowing each group to discuss whatever they choose, or a “*jigsaw*” -style activity where each group is given one question to focus on
 - a “*think-pair-share*” activity
 - a structured activity to simulate or recreate something discussed in the material to allow students to gain a deeper appreciation of the topic
 - as an example, after a reading or video on computer security, students could play \ (or at least read about and consider\) the game [_Control-Alt-Hack](#)

1. Debrief

- Ask one or more students to summarize the topics covered in the lesson and their thought or opinions
- Consider collecting some evidence of the activity, such as students' responses to the guiding questions or notes from small-group discussions, to evaluate engagement with the lesson

##BJC Lecture Suggestion

1.4.0.1 Good for Classroom Instruction/Background Information for Instructors

- [BJC Lecture 10: Social Implications of Computing](#)

* META: Computers in Education \(\text{Implications of Multiple Choice Tests}\) 0:00-4:30 * Computers in Education \(\text{Open?}\) Judah Schwartz 4:31 — * Tools 4:50-5:30 * Microworld 5:30-6:30 * Microworld Example Physics

- [BJC Lecture 12:Social Implications II Dr. Gerald Friedland](#) *Good for Classroom Instruction-Suggest Previewing due to Social Media Examples.

- Dr. Gerald Friedland Sr. Research Scientist at International Computer Science Institute \(ICSI\) on Sharing Multimedia and the Impact on Online Privacy\) 0:00-1:45
- Introduction to Social Media: The Price of Social Media Use-Stephen Colbert 1:50-6:25
- Observations on Sharing Data and Ineffective Privacy Protection 6:30-7:50
- Social Cause: Collection of Data Across Sites 7:50-10:30
- Multimedia in Internet is Growing 10:35-12:05
- CS Problem: Higher Demand for Retrieval and Organization of Data 12:07-13:05
- Manual Tagging & Geo Tagging 13:05-17:30
- Issue of Tracking & Dangers of Oversharing 17:30-18:31
- Berkeley Multimedia Location Estimation Project 18:31—
- ICSI's Evaluation Results 19:49
- YouTube Cybercasing 20:47
- Privacy Implication of Internet and Data 22: 30-25:40
- Person Linking Using Internet Videos 25:45-26:45
- Solutions for Privacy that Don't Work: Think Before You Post! 26:45-28:14

1.5 Accommodation\ /Differentiation

- Be aware of unconscious assumptions of context when presenting lessons relating to societal or culture topics. This is especially important in highly multi-cultural classrooms. When in doubt, ask questions or give brief explanations of any terminology, concepts, or ideas that are needed to appreciate the material.
 - For example, if showing “[Smartest Machine on Earth](#)”, be aware that some students may not be familiar with *Jeopardy!*.
- Feel free to adjust the pacing guide liberally to meet the needs of your chosen material. If a video or reading will require more than 20-25 minutes, consider splitting the lesson across two days.
 - Day one should include introduction of the topic, the video\reading, and a brief reflection, with the discussion or activity pushed to day two.
- Try to vary the topics of culture days throughout the semester to engage a broad range of students' interests and experiences. Not all students will connect with every lesson, but you should strive to have every student connect with at least one or two culture days each semester.

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Culture Day Lesson B: Student Research Project/Presentation

1.1 Learning Objectives

Students will be able to... * Describe in detail the topic of their assigned/chosen computer science related topic
* Answer questions about their topic * Explore and analyze the impact on, and impact of, technology in the context of their topic

1.1.0.1 Emphasize with students...

1.1.0.2 Curriculum Competencies - Applied Technologies

Coding is simply a technical skill. Yet the process of software development, product creation, and deployment, involves the intersection of diverse technical innovations, global community of peoples, cultural beliefs, values, and ethical positions. There is impact from, and impact on, our life and society at many levels (personal, community, global, and environmental), including the unintended negative consequences of the technology choices we make. These connections and discussions, help us appreciate the past, be wiser in the present, and more ready to embrace the future.

1.2 Possible Topics

- Famous figures in computer science (Donald Knuth, Alan Turing, Kernighan & Ritchie, Mark Zuckerberg, Bill Gates, Elon Musk, Steve Jobs, etc.)
- Famous women figures in computer science (Grace Hopper, Bletchley code breakers, Ada Lovelace, Dorothy Vaughan, Anita Borg, etc)
- Important technologies or algorithms (RSA, Dijkstra's Algorithm, RAID, integrated circuits, etc.)
- New and emerging technologies (AI, Machine Learning, robotics, cryptocurrencies, etc.)
- Impact of technology on society (social media, health and lifestyle, screen time, etc.)
- Ethics (privacy, cyberbullying, security, etc.)
- Legal issues from new technology (intellectual property, facial recognition discrimination, etc.)
- Software reliability and limitation (<https://www.cigniti.com/blog/37-software-failures-inadequate-software-testing/>)

This list should be expanded and updated, from time to time, to be up to date.

1.3 Materials/Preparation

- A list of possible topics for research projects
 - Encourage students to research from online and other resources, and keep track of sources

- Citation generator <http://www.easybib.com/k> this is a handy way to generate citations (even for websites) that can be included as a Bibliography or References for the project
- Guidelines for projects and/or presentations
 - Presentation Tips <https://www.thinkoutsidetheslide.com/top-5-powerpoint-tips-for-student-presentations-in-school/>
 - Ideas for giving interesting presentations <https://www.powtoon.com/blog/17-killer-presentations-tips-students-stand/>

1.4 Pacing Guide

| Duration | Description |
|------------|---|
| 5 minutes | Welcome, attendance, bell work, announcements |
| 15 minutes | Presentation #1 |
| 15 minutes | Presentation #2 |
| 15 minutes | Presentation #3 |
| 5 minutes | Debrief and wrap-up |

1.5 Instructor's Notes

1. *Prior to Culture Day*

- Assign each student one or more topics to research and present to the class on a future day
 - Topics can be assigned, chosen by students from a pre-defined list, or suggested by students and approved by instructors
- Create a schedule of when culture days will occur and which students will present on each day
- Depending on how many students are in the class, and how many days you wish to allot for presentations, your pacing guide can be adjusted.

2. Student presentations

- Each student should give a 5-7 minute presentation on their assigned topic, followed by 8-10 minutes for Q&A
 - Students should have a visual aspect to their project (poster, PowerPoint, prop bag, etc.) as well as giving a verbal presentation
 - Use your judgement regarding the level of technical detail expected in the presentation. It is probably not realistic to expect students to become experts in advanced technologies such as RSA, but they should be able to explain, at least at a high level, the details of their topic.
 - Do not allow students to simply read a textbook or online definition of the topic—ensure they can at least explain the subject in their own words.
 - Allow classmates to ask questions, but beware of students trying to stump each other.
 - Have a few questions for each assigned topic prepared ahead of time for instructors to ask in case classmates do not have questions.

1.6 Accommodation/Differentiation

- In smaller classes, each student may be able to present twice in a single semester.
- For classes where students are less experienced with presentations, consider a “science fair”-style event where students produce a display that can be viewed by others to present their topic.

TEALS Program

[Home](#) | [Curriculum Map](#)

1 Culture Day Lesson D: Interview with People in Technology

1.1 Learning Objectives

Students will be able to... * Identify the people in their family, or community, who have technology-related jobs
* Create interview questions to find out about their jobs * Conduct an interview * Identify the different types of roles and skills needed in the technology industry * Identify technology skills needed in non-technology organizations * Identify how **computational thinking** is practiced in real life * Reflect on how technology could be part of their future careers

1.1.0.1 Emphasize with students...

1.1.0.2 Curriculum Competencies - Applied Skills and Technology

Students may come with stereotypes and myths about technology jobs.

There are many different types of roles and skills needed in the technology industry. Not everyone is a coder; diffand not everyone is coding all day long. There are artistic designers, music creators, script writers, project managers, marketing specialists, quality assurance testers, front-end designer, etc.

As students get to know people in their community, they may even discover individuals who are willing to be mentors for your classroom in the days ahead.

1.1.0.3 Content - Computational thinking skills applied to various aspects of design

The four stages of computational thinking are: decomposition, pattern recognition, abstraction, algorithm design. These are bigs words! These steps appear in our daily life.

Ask experts who are working with the computer in your jobs about how these steps are applied. Ask them for specific answers. You may be surprised, and hopefully, inspired.

1.2 Materials/Preparation

- General tips for interviewing others that might help: <http://provisional.com/employers/employer-interviewing-tips>
- Computational thinking basics https://en.wikipedia.org/wiki/Computational_thinking

1.3 Pacing Guide

This lesson can be done over a period of 2 classes.

In the first class, students prepare for the interviews. This class could be combined with a Journal Writing day nicely (see Culture Day C). Or, it could be combined with a regular project day, where students focus on working on a specific project.

| Duration | Description |
|------------|---|
| 5 minutes | Welcome, attendance, bell work, announcements |
| 15 minutes | Introduce lesson and warmup activity |
| 15 minutes | Prepare interview questions |
| 10 minutes | Other activity (suggestion: Journal Writing) |
| 5 minutes | Debrief and wrap-up |

In the second class, students share and report on the interviews.

| Duration | Description |
|------------|--|
| 5 minutes | Welcome, attendance, bell work, announcements |
| 45 minutes | Each group presents their interview highlights and key reflections |
| 5 minutes | Debrief and wrap-up |

1.4 Instructor's Notes

1. Introduction to the lesson, and warmup activity

- Introduce the lesson to students
- Identifying local industry
 - Before students start thinking about who to interview, find out how much your students know about technology companies in your city or town. Spend some time to search online. Write on the board some examples of technology companies, or companies that have technology departments.
- Identifying people in your lives
 - Ask each student alone, or with partner, to brainstorm on a piece of paper 3-4 people who have technology-related jobs. Encourage students to think of a different variety of jobs.
 - Encourage students to think of people that they know personally.
- Share and decide
 - Find a partner, and share your list with each other.
 - Agree on 2 people from your combined list to interview together

2. Preparing for the interview

- In your pairs, prepare a list of interview questions for each person (questions may be different)
- Encourage students to think about the different people around them:
 - those who work in technology organizations
 - those who have technical positions in other types of organizations

- could be friend, relative, or even someone that you can approach
- Encourage students to create a variety of questions, for example:
 - what does your typical work day look like?
 - what do you like, or not like, about your job?
 - what inspires you?
 - what are some challenges?
 - what educational background is needed to do your job?
 - what kind of technical skills are needed in your job?
 - what advice do you have for students?
- Include questions related to computational thinking:
 - can you give an example of how “decomposition” is used in the context of work?
 - can you give an example of how “pattern recognition” is used in the context of work?
 - can you give an example of how “abstraction” is used in the context of work?
 - can you give an example of how “algorithms” is used in the context of work?

3. Conducting the interview

- Each pair should make a plan of how to conduct the interview in the coming week.
- This should be done outside of class time.
 - Remind students to be professional and respectful when interviewing
 - Contact the person first to arrange a time and place to meet
 - Be punctual; if working with a partner, remember to introduce all parties involved.
 - Tell interviewee the purpose of the interview is for you (students) to learn more about technology related careers
 - Ask politely for permission to share some of their answers (within the classroom)
 - Avoid questions that may be awkward or too personal (like salary), and always thank the person for their time.
 - Take some notes, while still listening and being attentive
- After the interview talk with each other about key points, and personal take-aways
 - What surprised you?
 - What was interesting?
 - What did you learn?
- Did you feel inspired or get ideas about possibilities jobs you could do in the future?
- Together, write 1-2 paragraphs of reflection
- Together, prepare a few powerpoint slides (or photos, brochures, or grab bag items) to capture your interview findings and reflections, which to use for sharing with peers.

4. Share with peers

- Ask each pair to share highlights, and key reflections, with the class.
- This can be done in one class period. The presentation time per group should be adjusted so to accommodate all students, so that no one feels left out.
- As a guideline for a 5-6 minute presentation, they can prepare 2-3 slides for each person interviewed.
- Students can also play short video clips from their interview as well.

1.5 Accommodation/Differentiation

- Students who have little experience with interviews may practice interviewing one another in the class with a few simple questions.
- Some students for a variety of possible reasons, may struggle to think of others to interview. They may be encouraged to consider teacher or staff in the school as potential interviewees. Be sensitive to assist students who are new to the community, those not living with family, those who don't speak English at home, or have other situations that they may feel embarrassed about.
- Working with a partner may alleviate some awkwardness for students who really can't think of anyone to interview.
- Working with a partner may alleviate some possible anxiety related to a face-to-face interview.
- Motivated students could video the interview as well, and show clips during peer sharing time, with permission from the interviewee. The final sharing could be done in the format of a prepared video.

TEALS Program

[Home](#) | [Curriculum Map](#)

1 AP Computer Science A Curriculum Development

This project contains the source for the TEALS AP Computer Science A Curriculum. Content can be browsed in the following ways:

- [On GitBook](#) — The official source for the book
- [On GitHub](#) — From the repository contents on GitHub
- [Locally](#) — From a local clone of the development repository

1.1 Style Guidelines

Please read the [Style Guidelines](#) before modifying the contents of this repo. They're short and ensure consistency across the docset.

1.2 Authors

- Christine Keefe (Curriculum Developer)
- Nathaniel Granor (TEALS Regional Lead)

1.3 License

This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International](#) license. See [LICENSE.md.-pdf.html](#) for the full license.

1.4 Acknowledgements

1.4.1 TEALS Summer Fellows

- Ben Watsky
- Julian Boss

1.4.2 Markdown Conversion & Repo Setup

- [Steve Hollasch](#)
- Kenney Chan

1.4.3 Special Thanks

- Glenn Durfee
- Peter Durham

1.4.4 AP CS Curriculum Squad Volunteers

- Kevin Wilson
- Leo Franchi
- Miki Friedman
- Jim Steinberger
- Robyn Moscovitz
- Eric Halsey
- Kevin Trotter
- Andrew Smith
- Paul Roales
- David Broman
- Yael Elmatad
- Glenn Durfee
- Peter Durham
- Nelson Collin
- Ralph Case
- Charley Williams
- Jeffrey Booth

1.4.5 AP CS Curriculum Squad Teachers

- Nate Binz
- Janet Roberts
- Brett Wortzman
- Ingrid Roche

formatted by Markdeep_1.093 

TEALS Program

[Home](#) | [Curriculum Map](#)

1 TEALS AP CS A Curriculum — How to Contribute

1.1 Repository Location

The curriculum's source code is hosted on Github at <https://github.com/TEALSK12/apcsa-public>. Restricted instructor-only content is hosted at <https://github.com/TEALSK12/apcsa-instructor>.

Did you catch an error in the TEALS AP CS A curriculum, or perhaps have an idea for an addition that would make it better? If so, we'd love to see what you've got!

1.2 Methods of Contributing

We can accept changes and suggestions to this repo in a bunch of different ways, so here they are from least involved to most.

1.2.1 Casual Communication

You can make comments via email to [Kenney Chan](#). If you have a GitHub account, though, it's better to submit an issue, as there's a higher guarantee that it won't get lost.

1.2.2 Submit an Issue

The best way to make sure your feedback is recognized, tracked, and handled is to [submit an issue](#) on GitHub. You'll need a [GitHub account](#) to do this.

1.2.3 Submit a Pull Request (PR)

This is the most involved route, but more powerful. It works for all kinds of issues, from fixing typos to making radical changes in curriculum. Of course, if you decide on something massive, make sure to vet the idea with us first.

If you're doing something small or obvious, and you're comfortable, feel free to just go straight to a pull request. For anything else, you might want to first [submit an issue](#). In the issue, mention that you're willing to do the work yourself. If the idea is sound, we'll give you the green light before you commit any effort or time.

1.3 Previewing Locally

To view this GitBook locally for previewing changes, first install the gitbook-cli tool:

```
npm install gitbook-cli -g
```

Then from the repository root run:

```
gitbook serve
```

For complete instructions, see the [GitBook website](#).

> Note: this method of previewing won't contain all the TEALS styling on the main site, but will give you a rough approximation before making a submission.

1.4 How to Submit a Pull Request (Advanced)

The following steps outline the easiest way to submit a PR:

1. Create a local working clone of the apcsa repository. If you haven't done this before, just go to the [main code page](#) and hit the "Clone or download" button. Explaining how to use Git is outside the scope of this page, so you'll need to know how to do this already.
2. Create and checkout a feature branch for your work. (The `master` branch is protected, and you won't be able to submit changes there.)
3. Make your changes in the feature branch. Make sure you follow the [style guidelines](#) as you do so.
4. Push your branch up to the GitHub repo. For example, if you work on the command line, and your feature branch is named "moar-glitter", then you'd do this:

```
git push origin --set-upstream moar-glitter
```

This just needs to happen once to establish the connection between your local branch and a branch of the same name up on GitHub. After that, if you have more changes to add, you can just `git push` while working in your feature branch.

5. When you're done and ready for review (and all of your commits have been pushed), head to the [main code page](#). There you should see your branch in a highlighted box with a "Compare & pull request" button on the right. If not, you can just hit the "New pull request" button. If you did the latter, there will be two buttons to control which branch is going to merge into which. In this case, the base branch should be `master`, and the compare branch should be `moar-glitter`. Once that's set, be sure to enter a descriptive comment about the pull request. If there's an associated issue, include the issue number, prefixed with a `#` character. For example, if this is a fix for issue 137, include `#137` in the comments.
6. Finally, hit the "Create pull request" button. This will automatically notify the reviewers. Several things could happen:
 - A reviewer approves your PR and merges it into `master`. Gitbook will see the changes and automatically build an update with your changes within a few minutes.
 - A reviewer rejects your PR. This shouldn't happen if you got approval in a submitted issue. If not, you should get a clear explanation why the change was not accepted.
 - A reviewer may point out problems with your PR, and request changes before approving. You'll need to monitor your PR to catch this. One aid is that GitHub tracks notifications in the upper right corner of the web pages (the bell icon), and should alert you to change requests.
7. After the dust has settled, *please delete your feature branch* from the GitHub repo. You may of course choose to keep it in any local clones, but this will keep the primary repo clean of branch clutter. The one exception to this is if your feature is long-running, and you plan to issue a string of PRs as work progresses.

Hopefully, all went well, and you've helped to make our curriculum even better — *thank you!*

