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.

Accessing the Curriculum

The AP Computer Science A Curriculum GitBook is located at https://www.gitbook.com/book/tealsk12/ap-computer-science-a/details.

The following is an Alpha release of the AP Computer Science A Curriculum in GitHub pages: https://tealsk12.github.io/apcsa-public/

For contributions to the curriculum, the AP Computer Science A GitHub repository is located at https://github.com/TEALSK12/

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.

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.

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:

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.

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.

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.

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.

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.

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.

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."

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
For a more general introduction to the Pokémon franchise, visit http://www.pokemon.com/.

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.

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.

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.

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 (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).

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.

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.

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

Digital Tools Associated with This Curriculum

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), like Eclipse. Students will need to be able to save and access their programming projects locally or in the cloud.

Integrated Design Environment (IDE) — Eclipse

Coding in Java requires the Java Development Kit and a text editor or IDE. There are many Java IDEs available. Currently most of the TEALS classrooms use Eclipse. Unit 1 includes directions for installing Eclipse.

Practice-It

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 it's IT policies.

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.

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.

Printing GitBook

The AP CS A GitBook can be printed by navigating to https://aka.ms/TEALSAPCSAPDF. 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"

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.