Microsoft

</> TEALS

Curriculum overview

Introduction

The *Introduction to Computer Science with MakeCode Arcade* curriculum is a flexible and approachable course adapted from the *TEALS Introduction to Computer Science* curriculum and built with Microsoft MakeCode Arcade as its core teaching platform. This is a course for a wide range of high school students from diverse backgrounds. The curriculum has its roots in *UC Berkeley CS 10*. The original TEALS course has been successfully implemented in hundreds of high schools.

Introduction to Computer Science with MakeCode Arcade is an engaging course that explores a variety of basic computational thinking and programming concepts through a project-based learning environment. Every unit

culminates in a comprehensive project and roughly 75% of student time is spent building projects and practicing the skills they are learning.

Figure 1. Link to introductory video on the TEALS program.

Visual and approachable

This course uses MakeCode Arcade, an approachable and visual programming system with a rich coding environment for students at any experience level. This course uses MakeCode Arcade’s block-based environment during the first semester, making it perfect for introducing students to coding for the first time. During the second semester, students transition to using a text-based language.

Document structure

This curriculum uses a flat folder layout for the curriculum documents and supporting files. Each unit contains an instructor overview document, which includes learning objectives, a pacing guide, CSTA mapping, and a word wall (glossary).

Within each unit, lessons are grouped into folders. Most lessons contain the following documents:

- **Bell ringer.** The bell ringer activity can be completed by the students as they enter the classroom. An accompanying slide is often provided in the lesson's slide deck that can be displayed as students arrive. Five to ten minutes at the start of class is typically allocated to the completion and discussion of the bell ringer activity.

- **Instructor guide.** The instructor guide provides learning objectives and a pacing guide for the lesson. The instructor guide also contains teaching tips, a list of solutions provided via the instructor resource site and other related material.
- **Slide deck.** Instructors can use the PowerPoint presentation to guide the conversation during the lesson. Many of the presentations contain scripts that instructors can use and modify as needed.
- **Student guide.** Students should have their own copies of the student guides, either printed or electronically. The students use the guides during the lab activities for hands-on practice with the skills developed during the lesson.

More information on the structure of lessons is provided later in the section titled *Daily lesson plans*.

Helping trios

While they typically work independently on their lab activities, students are encouraged to rely on their teams as a first line of support, before asking instructors for assistance. We call these groups *helping trios*, as groups of three seem to work best for this application.

Remote instruction

Most of the content can be adapted to remote instruction with little change, as most of the class time is devoted to independent work by the students. If your remote instruction platform supports breakout rooms, students can be placed into breakout rooms with their “helping trios” during the lab activities, so students can learn to rely on their teammates for assistance before requesting help from their instructors.

About this curriculum

Philosophy

This curriculum has been designed by the TEALS program to support computer science teachers and/or volunteer professionals teaching an introductory computer science course in a high school classroom. The curriculum was originally based on, and still borrows heavily from, the Beauty and Joy of Computing Curriculum (BJC) developed at the University of California, Berkeley. The TEALS curriculum has a heavier focus on the basic programming components of the course than BJC, sacrificing some of the advanced programming and conceptual topics that are less appropriate in an introductory high school classroom.

This curriculum advocates a "hands-on" learning approach in which students' primary means of learning is through discovery, experimentation, and

application. To that end, each unit is built around a large, culminating programming project that exercises the objectives of the unit. In addition, nearly all lessons in the curriculum include a guided activity of some kind to allow students to practice with and experience the concepts covered in the lesson first-hand. Taken together, the lessons provide the skills and support necessary to enable students to complete the project and demonstrate mastery of the unit's objectives. Substantial class time should be provided for the project in each unit to ensure students can demonstrate mastery of the skills from each unit before moving on.

Encouraging play

This curriculum encourages the use of "play." Students are encouraged to discover on their own, and the MakeCode environment is well-suited for such activities. It is unlikely that students will "break" the MakeCode environment, and they should be confident in their ability to work in the environment without fear.

When they encounter a problem, students should explore the environment and try to find a solution on their own (e.g., looking in the various drawers of the toolbox for an appropriate block or code snippet), rather than immediately raising a hand and requesting assistance from an instructor. Instructors should reinforce this behavior, guiding the students through the discovery process with appropriate questions. Instructors should applaud students who attempt to solve their problems with exploration and play, even if those attempts are unsuccessful.

Similarly, students who finish activities early should explore the environment on their own. In the Blocks environment, students can explore blocks that they have not previously used and attempt to understand how they function. In the typing environments (JavaScript and Python), students can explore the toolbox in a similar way, perhaps comparing the code snippets with their familiar Blocks counterparts. Students who learn something new should be encouraged to share their discoveries with their teammates and, perhaps, with the entire class. Programmers learn about new languages, features and frameworks in the same way; students should begin using and honing these skills early in their careers.

List of units

FIRST SEMESTER (BLOCKS)

0	Beginnings	Students learn about the course, classroom environment, algorithms, and MakeCode Arcade environment.
1	Sprites	Students learn the basics of the core entity in MakeCode Arcade: the sprite.
2	Event handlers and variables	Students learn about event handlers available in MakeCode Arcade. Students are also introduced to variables, strings, and decision structures.
3	Loops and arrays	Students learn about definite and indefinite loops and their application to arrays. Students are also introduced to searching within an array as well as frame-based animation.
4	Functions	Students learn to write functions as a problem-solving technique and to encourage code reuse.

5	Tile maps and platform games	In this optional unit, students learn about tilemaps and their myriad uses, including in platform games.
6	Capstone project	Students work in teams to create complex projects. Students also create marketing materials for their projects.

SECOND SEMESTER (JAVASCRIPT / PYTHON)

0	A return to Blocks	In this optional unit, students revisit the core concepts that they learned in the Blocks environment. This unit is intended for sessions that do not immediately follow a Blocks session.
1	Introducing JavaScript or Python	Students transition to a typing language, either Static Typescript (a.k.a. JavaScript) or Static Typed Python (simply called *Python* in the course materials). Variables and event handlers are revisited from the typing language perspective.
2	Core programming concepts	Students revisit core topics from units 2 and 3 in the Blocks course, extending their skills where appropriate. Students work with variables, decision structures, loops, arrays, and animations.
3	Functions	Students implement functions in their typing language. They also learn how to play simple melodies in their typing language.
4	Introduction to object oriented programming	Students learn very basic object-oriented programming and design techniques, focusing on subclasses that inherit from the Sprite class.
5	Tilemaps and platformers	In this optional unit, students create projects that leverage tilemaps, including platform games.

6	Advanced graphics	In this optional unit, students work with advanced graphics techniques, including parallax and mini maps.
7	Capstone project	Students work in teams to create complex projects. Students also create marketing materials for their projects.

Daily lesson plans

Most lesson plans in this curriculum are designed to represent a single 55- minute class period with average pacing. Each class will have slightly different needs, possibly including different period lengths, student capabilities, classroom interruptions, and more.

With a few exceptions, each lesson consists of the following components:

Welcome, announcements, and bell work

Five minutes are allotted at the beginning of each day for administrative tasks such as taking attendance, giving announcements, returning work, and other necessary actions. During this time, teachers are encouraged to assign “bell work” (called “bell ringer” activities) for students to work on.

- These activities aim to engage students with the subject immediately upon entering the room, and should be short, clear, and able to be completed by all students.
- Specific “bell ringer” activities are given in the lesson plans, but they should be chosen by the teacher to reinforce or preview specific topics with which students have struggled or are expected to struggle most.

Discussion & instruction

Most lessons begin with a brief period of instruction on the topic of the day. These sections should be kept as brief as possible—the primary means of student learning in most lessons will be the lab activities.

- The goal of the instruction section of the lesson is to motivate the concepts being exercised in the lab and provide a short demonstration to help students find the necessary parts of MakeCode Arcade.
- Teaching teams should vary the ways in which the instruction is

presented throughout the course, including class discussions, kinesthetic activities, demonstrations, Socratic seminars, occasional lectures, and other approaches.

Activity

The largest portion of time in each lesson is dedicated to a guided activity that allows students to explore and practice with the day's key topics. Each activity is broken down into several parts, each of which consists of several steps. In general, the steps in a single section build on each other, and each section covers a new topic or new application.

- It is intended that the labs be structured well enough for students to work on their own, but teachers should feel free to interject at appropriate points to assess student progress and provide additional guidance as necessary.
- On occasion, multi-part activities involve multiple cycles of instruction and activities.
- Many of the activities include extensions for advanced students who complete the lab early.

Debrief

After each activity has concluded, time is allotted for teachers to review and debrief the activities with students. In general, there is not enough time, nor is there necessarily the need, to go through the lab step by step. Students should be able to assess their own progress, at least partially, by verifying that their programs function as specified in the lab.

- Rather than presenting solutions to each step of the lab, teachers are encouraged to use the debrief time to focus on particularly tricky or noteworthy parts of the lab or to discuss areas where students struggled.
- Debrief time can also be used to compare different approaches to some of the problems, emphasizing that, in most cases, there is more than one valid solution.
- Whenever possible, use examples of student work rather than instructor-created solutions during the debrief—this is an excellent

chance to showcase students who solve problems in elegant, creative, or canonical ways.

Homework

This curriculum does not assign homework as part of its lessons. Because this curriculum is intended to be used in a wide variety of classrooms, some of which may include students that do not have regular access to Internet enabled computers at home, all work is done during class time. In some circumstances, assigning some lab activities as homework can enable the teaching team to regain in-class time for additional lessons or activities, but this must be done with care. If homework is assigned, then arrangements must be made so that any students who do not have the ability to complete the homework at home do not fall behind. Furthermore, it should be expected that some students will not complete the assigned homework, and teaching teams must have a way to both assess that homework was completed and ensure the material is reinforced briefly in class.

Quizzes

To gauge student understanding, unit quizzes have been added. These are intended as low stakes, formative assessments that allow students to revisit topics at the end of the unit to reinforce learning. They are open book, giving students incentive to take good notes. Ideally, the quizzes are not graded, and students would reflect on the answers they got wrong and learn from their mistakes. The quizzes and answer keys can be found with the protected materials for the course.

Grading

Student work consists of class participation, daily labs, end of unit projects, and final project.

Accommodations

Reading challenges and pair programming

To accommodate students with varied reading levels, consider combining "helping trios" with the concept of pair programming. In pair programming, developers work in pairs with distinct roles. One person, known as the *operator* or the *driver*, works at the computer, operating the keyboard and mouse and implementing the logic of the program. The other teammate, known as the *reviewer* or *observer*, considers the logic required to solve the problem at hand and guides the operator on how to craft the code. The reviewer also monitors the code as it is entered by the operator. Team members rotate roles frequently.

In our context, one student in the team acts as the operator while the other teammates are reviewers. The reviewers can read the relevant portion of the student guide aloud and consider code options, which the operator then implements. Students should rotate roles frequently, say every three to five minutes. You can use a timer to remind students to rotate roles.

Visually impaired students

We have attempted to make all documents as accessible as possible. MakeCode Arcade has a high contrast mode to assist users who are visually impaired. The code editor for JavaScript and Python is compatible with screen readers, although the Blocks editor is not.

Creative Commons Attribution Non-Commercial Share Alike License