SAMSUNG

SAMSUNG MOBILE
APP ACADEMY

Guide to Teaching Mobile App Development

CALCULATE > BUILD

Classroom Lesson Plans and Resources • Education Standards

To download all classroom resources, go to the Samsung Mobile App Academy microsite at **scholastic.com/samsungacademy/teachers**.

For additional resources, the "learn to code" tab can be accessed at **scholastic.com/samsungacademy/learntocode**.

Mobilize the Future!

SAMSUNG

SAMSUNG MOBILE
APP ACADEMY

Guide to Teaching Mobile App Development

CALCULATE > BUILD

Classroom Lesson Plans and Resources

Grades 9–12

Contents

SAMSUNG

SAMSUNG MOBILE
APP ACADEMY

Guide to Teaching Mobile App Development

CALCULATE > BUILD

Classroom Lesson Plans and Resources

Grades 9–12

WELCOME TO THE WORLD OF MOBILE APPS!

Use the Samsung Mobile App Academy classroom resources and make your classroom part of the mobile revolution! The overall objective of this Samsung Mobile App Academy **Guide to Teaching Mobile App Development: *Calculate* > *Build*** is to provide a resource for teaching high school students the basics of mobile application development. With the goal of expanding students' problem-solving knowledge, critical-thinking skills, and technical programming skills, the Samsung Mobile App Academy materials utilize four steps: ***Ideate* > *Plan* > *Calculate* > *Build***. This Guide, concentrating on the ***Calculate* > *Build*** steps and the coding aspects of developing an app, **requires technology and technical knowledge**.

Utilizing science, technology, engineering, and mathematics (STEM) standards, these lessons give students the foundation to help them cultivate necessary 21st-century skills—the ***Calculate* > *Build*** Guide offers technology-infused lessons with flexible scheduling and easy integration into classrooms and extracurricular activities. Paired with the classroom resources that are available on the Samsung Mobile App Academy website, the ***Calculate* > *Build*** Guide provides the tools required to teach mobile app coding to grades 9–12 in three coding levels: beginner, intermediate, and advanced.

To learn more about how to teach students to use their creativity to refine an app idea that **does not require technology**, please see the **Guide to Teaching Mobile App Development: *Ideate* > *Plan***, which is also available on the Samsung Mobile App Academy website.

Use the Samsung Mobile App Academy classroom materials to ignite students' interest, and make your classroom a part of the mobile revolution!

To download all classroom materials, go to the "Teachers" tab on the Samsung Mobile App Academy website at scholastic.com/samsungacademy/teachers. For more information about the Samsung Mobile App Academies, go to scholastic.com/samsungacademy.

Education Standards

The following charts connect Samsung Mobile App Academy student activities to the knowledge, skills, and practices required in science, technology, engineering, and math (STEM) disciplines.

EDUCATION STANDARDS

GRADES 9-12

Strand ID	CCSS	What Students Do
Reading	**Science & Technical Subjects**	**Samsung Mobile App Academy**
Key Ideas and Details	**RI.11-12.3.** Analyze a complex set of ideas or sequence of events and explain how specific individuals, ideas, or events interact and develop over the course of the text.	Follow complex instructions to learn to use App Inventor to build a mobile application. Work to code a mobile app based on client specifications. Test, evaluate, modify, and reflect on the app they have created.
Craft and Structure	**RI.11-12.4.** Determine the meaning of words and phrases as they are used in a text, including figurative, connotative, and technical meanings; analyze how an author uses and refines the meaning of a key term or terms over the course of a text (e.g., how Madison defines faction in Federalist No. 10).	Review industry mobile app–developer documentation format and terms.
Integration of Knowledge and Ideas	**RI.11-12.7.** Integrate and evaluate multiple sources of information presented in different media or formats (e.g., visually, quantitatively) as well as in words in order to address a question or solve a problem.	Read and comprehend technically challenging information from the field of mobile app development.
Writing	**Science & Technical Subjects**	
Text Types and Purposes	**W.11-12.1.A.** Introduce precise, knowledgeable claim(s), establish the significance of the claim(s), distinguish the claim(s) from alternate or opposing claims, and create an organization that logically sequences claim(s), counterclaims, reasons, and evidence.	During the lesson and afterward, write competitive analysis, purpose and vision summaries, and audience and user profiles for original mobile app concepts.
Writing	**Informational Texts**	
Production and Distribution of Writing	**W.11-12.4.** Produce clear and coherent writing in which the development, organization, and style are appropriate to task, purpose, and audience. **W.11-12.6.** Use technology, including the Internet, to produce, publish, and update individual or shared writing products in response to ongoing feedback, including new arguments or information.	Using Samsung tablet technology or paper, during the lesson and afterward, write an industry-style creative brief of an app concept to solve a specific assigned scenario. Using available technology, present a mobile app creative brief to others for feedback.

Source: National Governors Association Center for Best Practices, Council of Chief State School Officers Title: Common Core State Standards, Publisher: National Governors Association Center for Best Practices, Council of Chief State School Officers, Washington D.C. Copyright Date: 2015

Education Standards
(Continued)

EDUCATION STANDARDS

Speaking & Listening		
Comprehension and Collaboration	**SL.11-12.2.** Integrate multiple sources of information presented in diverse formats and media (e.g., visually, quantitatively, orally) in order to make informed decisions and solve problems, evaluating the credibility and accuracy of each source and noting any discrepancies among the data.	Gain, evaluate, and present complex information, ideas, and evidence specifically through speaking and listening.
Presentation of Knowledge and Ideas	**SL.11-12.5.** Make strategic use of digital media (e.g., textual, graphical, audio, visual, and interactive elements) in presentations to enhance understanding of findings, reasoning and evidence, and to add interest.	Present the concept for development of an original mobile app in spoken, visual, and written industry formats, assisted by tablet technology.
Language		
Vocabulary Acquisition and Use	**L.11-12.6.** Acquire and use accurately general academic and domain-specific words and phrases, sufficient for reading, writing, speaking, and listening at the college and career readiness level; demonstrate independence in gathering vocabulary knowledge when considering a word or phrase important to comprehension or expression.	Learn and use industry-specific terminology associated with mobile app development.
Math		
Mathematical Practice	**MP2.** Reason abstractly and quantitatively.	Interpret numerical and graphic depictions of the expanding mobile app market to understand creative and career opportunities.

Source: National Governors Association Center for Best Practices, Council of Chief State School Officers Title: Common Core State Standards, Publisher: National Governors Association Center for Best Practices, Council of Chief State School Officers, Washington D.C. Copyright Date: 2015

NEXT GENERATION SCIENCE STANDARDS (NGSS)

Science & Engineering	NGSS	What Students Do
Engineering Design	**HS-ETS1-2.** Design a solution to a complex real-world problem by breaking it down into smaller, more manageable problems that can be solved through engineering.	Create a solution in the form of a mobile app designed to address a community problem that an imaginary client has identified. Break down the app concept into specific functions that the app will perform and create the code necessary for the app to perform each of these functions.

Source: NGSS Lead States. 2015. Engineering and Design HS-ETS1-2. *Next Generation Science Standards: for States, By States.* Washington, D.C.: The National Academies Press.

The *Ideate > Plan > Calculate > Build* Process

Iterative Process Comparison

Teaching Tip

To effectively implement these lessons, it may be helpful to explore how the Ideate > Plan > Calculate > Build process for mobile app development is similar to problem-solving procedures that you and your students already know.

The chart below compares the iterative industry Ideate > Plan > Calculate > Build process to similar step-by-step cycles described for core education disciplines and standards. Depending on your course or situation, you may wish to share with your students the comparison to one or more columns.

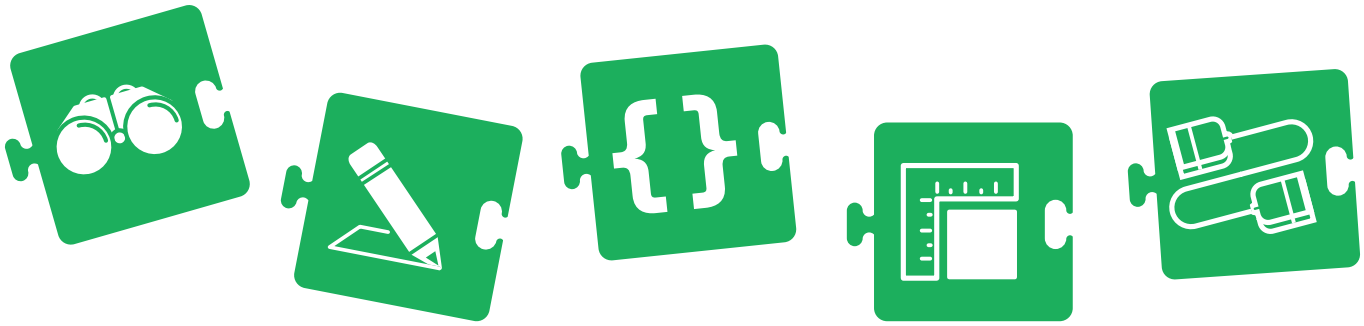
ITERATIVE PROBLEM-SOLVING PROCESSES COMPARED

Ideate > Plan	Scientific Inquiry[1]	Engineering Design[1]	Mathematical Modeling	Creative Writing
Discover everything you can. **Define** purpose and vision.	Formulate a question	Define a human need	Consider a real problem	Focus on a topic or scenario
Research competitors	Research how others have answered it	Research how others have solved it	Research existing models	Prewrite
Define your idea	Brainstorm hypotheses and choose one	Brainstorm solutions and select one	Make modeling assumptions	Organize
Design a rough sketch or concept	Conduct an experiment	Create and test a prototype	Set up a mathematical model	Draft
Calculate > Build	**Scientific Inquiry[1]**	**Engineering Design[1]**	**Mathematical Modeling**	**Creative Writing**
Develop solutions	Modify hypothesis based on results	Redesign solution based on tests	Analyze model	Review and get feedback
Build your codebase	Draw conclusion, write paper	Finalize design, make drawings	Interpret results and compare with reality	Revise and edit

1 Museum of Science. Boston. *Engineering the Future*™. Copyright © 2014. *It's About Time*®. All Rights Reserved.

The *Ideate > Plan > Calculate > Build* Process
Description

The ***Ideate > Plan > Calculate > Build*** process, as employed in workplace app development, is integral to the instructional sequence and the student support materials provided. The ***Ideate > Plan > Calculate > Build*** process guides students in a logical order and identifies the steps of a complex multistep procedure that helps students develop good habits for future problem solving:



Ideate > Plan

Discover everything you can

- Define purpose and vision
- Research competitors
- Who would use the app?

Define your idea

- Build User Profiles
- Build User Journeys
- Define features for your user profiles

Design a rough sketch or concept

- Create the primary screens
- Focus on interaction and data
- Remove friction from your app
- Build functional specifications and wireframes
- Test your concept with friends, family, and others

Calculate > Build

Develop solutions

- Design algorithms
- Map data and dependencies

Build your codebase

- Build features and screens
- Iterate and test as you build

Teaching Tip

To delve further into conceptualizing what goes into creating a mobile app, and that does NOT require technology, consult the **Guide to Teaching Mobile App Development: *Ideate > Plan***, which is also available under the "Teachers" tab on the Samsung Mobile App Academy website at **scholastic.com/samsungacademy/teachers**.

Integrating Technology
Preparing for the Lessons

The curriculum in the **Guide to Teaching Mobile App Development: *Calculate > Build*** requires technology and technical knowledge. The Guide provides students with an opportunity to work closely with technology and to learn skills to build an app. Although these lessons can easily be adapted and implemented as individual student projects, the Guide is written with the assumption the students are working in a team situation.

For teachers interested in lessons aimed at conceptualizing what goes into creating a mobile app and what does NOT require technology, the **Guide to Teaching Mobile App Development: *Ideate > Plan*** (also available for download under the "Teachers" tab of the Samsung Mobile App Academy website) is a perfect companion to the **Guide to Teaching Mobile App Development: *Calculate > Build*** coding lessons. Focusing on the practical and critical-thinking processes that developers use when conceptualizing and creating mobile apps, the ***Ideate > Plan* Guide** fits well after the ***Calculate > Build* Guide's** Lesson 1 or 2. However, **neither Guide is dependent on the other** so the lessons in each can be used independent of each other.

TECHNOLOGY REQUIRED

The lessons in the **Guide to Teaching Mobile App Development: *Calculate > Build*** rely on the use of technology. The Guide provides teachers with the necessary resources, plus instructions for use with students in the three levels of coding experience—beginner, intermediate, and advanced. It is very important to carefully prepare for these lessons, paying particular attention to the following:

- **Computers**
  One computer with Internet access should be available for every student team for Coding Challenges (available under the "Teachers" tab on the website).

- **Android Mobile Devices (optional)**
  If available, students can use Android mobile devices for Coding Challenges 1 and 3 (Documentation and Debugging, and Connected Services). If resources are not available, students can use MIT App Inventor Companion to test.

- **LCD Projector (optional)**
  Ideally, and if available, the teacher can project the ***Calculate > Build*** Classroom Presentation throughout the lessons and during student-facing discussions. A projector can also be used for the teams to project their apps during their presentations. If a projector is not available, teams can view the ***Calculate > Build*** Classroom Presentation and student apps on their computers or Android mobile devices.

- **MIT App Inventor***
  MIT App Inventor was created as an open-source project by Google, with MIT eventually taking the helm. Students will use the App Inventor program to complete Coding Challenges 1 and 3 (Documentation and Debugging, and Connected Services). It is essential that teachers are familiar with this program prior to beginning these Coding Challenges. An "App Inventor Overview" is available under the "Teachers" tab of the Samsung Mobile App Academy website at **scholastic.com/samsungacademy/teachers**.

- **MIT App Inventor Companion**
  If Android Mobile Devices are not available, students can use the MIT App Inventor Companion to test their applications using the available computers. For more information, please visit **http://appinventor.mit.edu/explore/ai2/setup.html**.

- **Greenfoot**
  Students will use Greenfoot to complete the Animation Coding Challenge.* It is essential that the program is installed and that teachers are familiar with this program prior to beginning the Coding Challenges. For more information about how to use Greenfoot and to download the software, please visit their website at **http://greenfoot.org**.

- **Scratch**
  Students will use Scratch to complete the Cryptography Coding Challenge.* It is essential that teachers are familiar with this program prior to beginning the Coding Challenges. For more information about how to use Scratch, please visit their website at **http://scratch.mit.edu**.

- **Twitter Account and API Key**
  Students will use Twitter to complete the Connected Services Coding Challenge.* To complete the Coding Challenge, teachers will need to provide (or have students create) a Twitter account and a Twitter development API key. For more information, please visit **https://twitter.com** (Twitter account) and **https://dev.twitter.com** (development API key).

* Materials available under the "Teachers" tab on the Samsung Mobile App Academy microsite: **scholastic.com/samsungacademy/teachers**.

CALCULATE > *BUILD* RESOURCE MATERIALS

All the resources necessary to feel confident teaching students how to code are available under the "Teachers" tab of the Samsung Mobile App Academy website at **scholastic.com/samsungacademy/teachers**. In particular, the website offers the following:

- **Guide to Teaching Mobile App Development: *Calculate* > *Build***
  Lesson plans are modular and adaptable. Activities enhance education standards and accommodate various schedules or STEM curriculum integration.
- ***Calculate* > *Build* Classroom Presentation**
  The student-facing Classroom Presentation, with slides that correlate to the specified sections in the Guide, is a useful visual and reference tool for both teachers and students.
- ***Calculate* > *Build* Coding Challenges**
  Available under the "Teachers" tab on the website in the "Student Materials" section, the four Coding Challenges correlate and accompany each lesson in this Guide. To accommodate all students, the Coding Challenges are available in beginner, intermediate, and advanced levels. **(Also see the Student Activity Levels section below.)**
- **MIT App Inventor Overview**
  Available under the "Teachers" tab on the website, the MIT App Inventor Overview is a detailed, illustrated tutorial demonstrating how to use MIT App Inventor. Students will use the App Inventor program to complete Coding Challenges 1 and 3 (Documentation and Debugging, and Connected Services).
- **Resources and Inspiration**
  A reference list of industry-vetted URLs for mobile app development news, blogs, inspiration, methods, design, and programming resources is located in this guide, or can be downloaded at: **scholastic.com/samsungacademy/resources**.

STUDENT ACTIVITY LEVELS

(also see *Calculate* > *Build* Coding Challenges description above) [Presentation Slide 3]

Student Coding Challenges (activities) are delivered in three levels: beginner, intermediate, and advanced. Before students start the Coding Challenges, have the students identify the level they feel best represents their existing coding skills:

BEGINNER	New to coding and needs to be introduced to the basics.
INTERMEDIATE	Familiar with a language (or two) and wants a challenge.
ADVANCED	Has the skills to build an app or website and wants to perfect or further refine the process.

SCHEDULING

The lessons in this Guide may be adapted to fit various schedules, such as 45-minute class periods, block periods or units, or a full day.

Possible formats for using Samsung Mobile App Academy classroom resources include:

- a module within a course in any STEM discipline
- a weekly club or after-school activity
- an intensive special-event weekend or summer academy experience

Teaching Tip

If in-classroom technology or technology time is limited, students might be instructed to use computers in the school library after school or to continue to work at home if they have a computer.

TEAMWORK

These lessons can be adapted and implemented as individual student projects or as team projects. Students can work independently or in teams of two students of similar coding experience levels—beginner, intermediate, or advanced. Working in teams allows for the project to mimic a real-world environment and may best allow for creative and knowledge exchanges.

Lesson Plan 1

ENGINEERING SOFTWARE

Calculate > Build Classroom Presentation Slides 4–27

Lesson Objective: Students will be able to articulate the difference between software engineering and coding. Students will be able to use search engines to find and understand coding language resources and documentation.

Materials: The ***Calculate > Build*** Classroom Presentation [Presentation Slides 4–27],* Documentation and Debugging Coding Challenge (beginner, intermediate, or advanced),* computers, App Inventor,* Android devices (optional), and App Inventor Overview*

* Materials available under the "Teachers" tab on the Samsung Mobile App Academy microsite: **scholastic.com/samsungacademy/teachers**.

Total Estimated Time: approximately three hours, four 45-minute class periods, or two 90-minute class periods (lessons can be adapted to various formats and schedules).

LESSON 1.1—WHAT DOES IT MEAN TO BE A SOFTWARE ENGINEER? [PRESENTATION SLIDES 5–10]

This lesson aims to provide students with an overview of the many roles a software developer must fill when building software.

- **Discussion Topic: What Does It Mean to Be a Good/Great Software Engineer?** [Presentation Slide 6] Pose the question to students and collect their answers. Ask them to list the steps they believe are necessary before code can be written. Do the students feel that code is the most important part of the process or are there other steps that might be more important?
- **Discussion Topic: Software Engineering Roles** [Presentation Slide 7] Explain that software engineers solve problems with software, plan and organize platform architecture, build algorithms for data manipulation, create documentation of their process and solutions, research and interpret documentation, support and manage developer communities, and write code. Pose the question that if software engineers are responsible for so many other tasks, how important is the code or the language?
- **Discussion Topic: Software Languages** [Presentation Slides 8–10] Explain to students that familiarity with a specific programming language can be great but as software engineers it is their job to know how and why things get built. Programming languages will evolve but the process of engineering problem solving will remain. Talk about the history and applications of the Java language. Why did one language change so often? What kept it strong?

Teaching Tidbits

Java became popular because it allowed developers to create an application with one language for multiple operating systems—"write once, run anywhere" (WORA). In terms of syntax, Java was advanced compared to other major languages of its time, and when browsers started adapting applets, Java took off. Java inspired many languages such as C#, which was dubbed to be "as easy as Java but as powerful as C++." Java was a "jack of all trades," but eventually other specialized technology started overtaking it. Java's flexibility, however, has allowed it to survive.

LESSON 1.2—DOCUMENTATION AND SUPPORT [PRESENTATION SLIDES 11–16]

This lesson aims to provide students with an overview on a programming language's resources and documentation and how it can be used in software engineering. [Presentation Slide 12]

- **Discussion Topic: The Benefit of Using an SDK** [Presentation Slides 13–16] Ask students to identify why the Android SDK is a useful tool for developing mobile software. Ask them to identify the kinds of features the Android SDK allows developers to access out of the box. List features such as: manage app distribution, payment processing, account management, training, support, community, and documentation. Explain that programming languages, like Java, have persisted because they had communities behind them that support their development.

LESSON 1.3—DEBUGGING [PRESENTATION SLIDES 17–23]

This lesson aims to help students understand common programming mistakes and provide them with a basic troubleshooting work flow for when their codebase is not operating as expected.

- **Discussion Topic: What Is Debugging?** [Presentation Slide 18] Explain to students that debugging is the process of finding and removing bugs, errors, and abnormalities from code that would otherwise cause unexpected behaviors in software.

Teaching Tidbits

The engineer who coined the term "debugging" was Grace Hopper in 1947. While programming, she discovered a moth inside her computer that was causing operational issues. To get things running correctly she had to "de-bug" (remove the moth). She also invented the world's first compiler.

- **Discussion Topic: Debugging Your Software** [Presentation Slide 19] Explain to students that even the most seasoned developers introduce bugs into their code as a normal part of the programming process (mistakes happen). When dealing with a program that is not operating correctly it is important to follow a basic process of: replicate the problem, analyze the error, locate the section of your code that is causing the unexpected behavior, and inspect the situation to decide which debugging tool is right for the job. If you cannot fix the problem, get help from someone else to review with a new perspective.

▶ **Discussion Topic: Common Mistakes**
[Presentation Slide 20] Review the list of common code mistakes typically found when building software, which may include incorrect spelling, missing variables, unused blocks of code, incorrect logic, wrong function blocks, missing logic, missing code in a function, and string concatenation.

▶ **Discussion Topic: The Process**
[Presentation Slides 21–23] Explain that programming or writing code isn't about building a perfect codebase or following all of the rules—it's about creatively executing a plan and learning how to keep moving forward when you get stuck. When programming, students should keep the following topics in mind:

- Mistakes happen! The best programmers know that mistakes are part of the process. Don't hesitate to try something new and don't get upset when you mess up.
- Check your resources! Don't be afraid to look things up. Use documentation, Stack Overflow, and Google to see if you can figure out the answer to a problem before you ask someone else.
- Feeling stuck? Double-check your dependencies, look for typos, give your mom a call. Often, taking a few minutes to get away from a problem is all you need in order to solve it.

▶ **Student Activity: Documentation & Debugging Workshop**
[Presentation Slides 24–26]

Student Activity Objective: Students will work to test, identify, and repair bugs in an existing codebase in order to get the Flashcards application working correctly.

Instructions: Using the Documentation and Debugging Coding Challenge level appropriate to your student teams, have students work as individuals or pairs to complete the Documentation and Debugging Coding Challenge. The activity should be introduced after the above Lesson 1 classroom instruction and discussions have been completed.

Student Activity Tips

- When individual students or pairs get stuck on a bug or an issue, have students review their progress with a student from the same activity level to assist in discovering the issue.
- Before jumping into the code, students should use the fully functioning application to catalog and document all of the application's features. You may have students install the .apk file on Android devices in class (if available) or make a single Android device available to all students to use throughout the activity.
- Remind students that search engines and documentation can be a powerful tool when repairing or debugging code.
- For best results, require students to plan their solution (using pencil and paper) or document their dependencies and issues before beginning to code. Activity levels can be scaled up or down depending on a student's ability.

Lesson Plan 2

Lesson Objective: Students will be able to identify the different kinds of animation used in mobile applications by deconstructing and understanding how they are built. .

Materials: The ***Calculate > Build*** Classroom Presentation [Presentation Slides 28–63],* Animation Coding Challenge (beginner, intermediate, advanced),* computers, Greenfoot (install—see URL listed in the **Integrating Technology—Preparing for the Lessons** section of this guide)

* Materials available under the "Teachers" tab on the Samsung Mobile App Academy microsite: **scholastic.com /samsungacademy/teachers**.

Total Estimated Time: approximately three hours, four 45-minute class periods, or two 90-minute class periods (lessons can be adapted to various formats and schedules).

LESSON 2.1—WHAT IS ANIMATION? [PRESENTATION SLIDES 29–60]

This lesson aims to provide students with an overview of the four most popular ways to animate objects in applications, and generally explains how they are built.

- **Discussion Topic: What Is Animation?** [Presentation Slides 29–30] Pose this question to students and collect their answers. Explain to students that any time a digital object moves inside an application it is being animated. Explain to students that animations are an important part of application development and can be used to hide or display content, give visual cues for interaction, or translate real-life experience inside digital spaces. Ask students to provide examples of animations found in the applications they use most.

- **Discussion Topic: Basic Animations** [Presentation Slides 31–36] Explain to students that basic animations can be categorized as any frame-based animation. Frame-based animations follow a defined path for animation and do not react to hardware or environmental conditions. Explain to students that basic animations have a beginning and an endpoint and are programmed to either start or stop. Ask students to provide examples of common basic animations. Review the example on Presentation Slide 36 to provide context and ask students to find additional examples.

- **Discussion Topic: Motion Animations** [Presentation Slides 37–44] Explain to students that motion animations are animation effects that react to device hardware. Ask students to list the kinds of hardware features that might be used to trigger a software animation. Examples may include GPSs, accelerometers, gyroscopes, and microphones. Explain to students that popular features such as Augmented Reality or Parallax are built using motion animations. Review the example materials with students to provide additional context and ask students to find additional examples.

- **Discussion Topic: Sprite Animations** [Presentation Slides 45–53] Explain to students that sprite animations are the most ubiquitous of all the animation types and are used heavily in games, websites, and mobile applications. Describe to students that sprite animations are built using a single image file and the viewable area of the graphic is switched programmatically. Review provided sprite sheets with students and describe how a single sprite sheet can be used to animate a complete game. Ask students to use search engines to search and find sprite sheets for games they play and show them to the class.

- **Discussion Topic: Physics Animations** [Presentation Slides 54–60] Explain to students that advances in hardware have resulted in the ability of programmers to build digital worlds with physical properties. Physics engines, responsible for creating physics animations, allow developers to create digital environments with distinct properties such as friction, gravity, elasticity, speed, and direction. Explain to students that physics engines make it possible to build games like Angry Birds. Review the example materials with students to provide additional context and ask students to think of other examples of games or animations that use physics engines.

- **Student Activity: Animation Workshop** [Presentation Slides 61–63]

  Student Activity Objective: Students will work to plan, code, and configure an animation that can be controlled using a computer keyboard.

  Instructions: Have students work as individuals or pairs to complete the Animation Coding Challenge using the appropriate Animation Coding Challenge level. The activity should be introduced after the Lesson 2 classroom instruction and discussions have been completed.

Student Activity Tips

- Students can work individually or in pairs on this activity. When they get stuck on a bug or an issue, have them review their progress with a student from the same activity level.
- Remind students that search engines and documentation can be a powerful tool when repairing or debugging code.
- For best results, require students to plan their solution (using pencil and paper) or document their dependencies and issues before beginning to code. Activity levels can be scaled up or down depending on a student's ability.

Lesson Plan 3

Calculate > Build Classroom Presentation Slides 64–74

Lesson Objective: Students will be able to articulate the benefits and disadvantages of developing an application using connected services.

Materials: The ***Calculate > Build*** Classroom Presentation [Presentation Slides 64–74],* Connected Services Coding Challenge (beginner, intermediate, or advanced),* computers, Android devices (optional), App Inventor Overview,* Twitter account, Twitter API key

* Materials available under the "Teachers" tab on the Samsung Mobile App Academy microsite: **scholastic.com/samsungacademy/teachers**.

Total Estimated Time: approximately three hours, four 45-minute class periods, or two 90-minute class periods (lessons can be adapted to various formats and schedules).

LESSON 3.1—WHAT ARE CONNECTED SERVICES? [PRESENTATION SLIDES 65–71]

This lesson aims to provide students with an overview of what connected services are, how they operate, and the long-term benefits and disadvantages of building with them.

- **Discussion Topic: What Are Connected Services?** [Presentation Slide 66] Explain to students that connected services offer developers the ability to access resources and features that might otherwise be too difficult or expensive to build. By using an API (see Key Terms in this guide), companies like Google, Facebook, and Dropbox allow developers to connect to their core infrastructure and scale features quickly with very little coding.

- **Discussion Topic: APIs vs. SDKs** [Presentation Slide 67] Explain to students that APIs and platform SDKs (see Key Terms in this guide) often go hand in hand. An SDK is typically only a wrapper that makes it easy to access a platform's API. For example, Facebook has an Android SDK that integrates Facebook login, sharing, etc. An API is an idea while an SDK ships with your code. Ask students to list popular APIs or SDKs and then discuss how they work together.

- **Discussion Topic: Connected Service Features** [Presentation Slide 68] Ask students to provide examples of the kinds of services connected platforms or APIs can provide to developers. Go through the following list and explain to students why each of the features might be too large or too expensive for most developers to build: advertising, analytics, app invitations, authentication, social sharing, monetization/payments, messaging, data, maps/geolocation.

- **Discussion Topic: Why Do Connected Services Produce APIs/SDKs?** [Presentation Slide 69] Explain to students that companies like Facebook and Google benefit from building and managing their connected service APIs. Discuss that APIs can benefit from the connected service relationship in several ways. Review these common examples:
  - charge developers or businesses for access
  - increase user base by requiring accounts
  - allow write privileges that expand collected data
  - promote their brand with in-app logos or links
  - control the future and philosophy of a feature through market ownership

- **Discussion Topic: How Do Connected Services Work?** [Presentation Slide 70] Explain to students that using a connected service's API requires you to create and maintain a verified identity that is essentially a digital handshake. While services may allow you to download and access content from their system it is almost always (with a few rare exceptions) still owned by that platform and will need to be deleted or destroyed when the handshake is broken. It is also important for students to understand that accessing and making content available costs companies money, so often these digital, handshake relationships have limits to the kinds and quantity of information they will allow a developer to access in a given day or week.

- **Discussion Topic: Building With Dependencies** [Presentation Slide 71] Explain to students that connected services can go a long way in helping developers build robust software but they also come with some known liabilities. As developers, your access to an API is limited to the terms of the service and connected services should be considered a privilege and not a right. Remind students that working with a connected service is a platform dependency and that the following topics need to be understood before programming can begin: data limitations, licensing, terms and conditions, access and availability.

- **Student Activity: Connected Services Workshop** [Presentation Slides 72–74]

  Student Activity Objective: Students will work to plan and code an application that uses the Twitter API to post and retrieve tweets to a Twitter account.

  Instructions: Have students work as individuals or pairs to complete the Animation Coding Challenge using the appropriate Animation Coding Challenge level. The activity should be introduced after the Lesson 3 classroom instruction and discussions have been completed.

Student Activity Tips

- You can have the Twitter account and API key ready before class or you can demo how to log in to the Twitter Developer portal and register an API key for their application and require each student to use a unique one.
- Students can work individually or in pairs on this activity. When they get stuck on a bug or an issue, have them review their progress with a student from the same activity level.
- Remind students that search engines and documentation can be a powerful tool when repairing or debugging code.
- For best results, require students to plan their solution (using pencil and paper) or document their dependencies and issues before beginning to code. Activity levels can be scaled up or down depending on a student's ability.

Lesson Plan 4

Calculate > Build Classroom Presentation Slides 75–88

Lesson Objective: Students will be able to articulate how cryptography works. Students will be able to compare the benefits and disadvantages of encryption and hashing.

Materials: The ***Calculate > Build*** Classroom Presentation [Presentation Slides 75–88],* Encryption Coding Challenge (beginner, intermediate, or advanced),* computers, Scratch (see URL listed in the **Integrating Technology—Preparing for the Lessons** section of this Guide)

* Materials available under the "Teachers" tab on the Samsung Mobile App Academy microsite: **scholastic.com/samsungacademy/teachers**.

Total Estimated Time: approximately three hours, four 45-minute class periods, or two 90-minute class periods (lessons can be adapted to various formats and schedules)

LESSON 4.1—CRYPTOGRAPHY BASICS [PRESENTATION SLIDES 76–88]

This lesson aims to provide students with an overview of the role of cryptography and explain the benefits and disadvantages of encryption and hashing.

- **Discussion Topic: What Is Cryptography?** [Presentation Slide 77] Explain to students that the art of obfuscating content (cryptography) is thousands of years old. Ask students to discuss why it is important to protect digital content. Ask students to list examples of instances and types of data that must be protected. Bring up the following examples if students do not mention them: usernames/passwords (authentication), fingerprints/biometrics (authentication), credit cards, emails (encrypted in transmission), SSL (Secure Sockets Layer).
- **Discussion Topic: Applying Cryptography** [Presentation Slide 78] Explain to students that cryptography can be applied in two basic ways:

Teaching Tip

The access/data analogy does not represent all of the subtleties associated with cryptography and digital security. It is intentionally simplified to give students an analogy they can grasp easily.

  - Access: illustrated as the guard post on a street before a building. It's the ability to get to your system but does not apply (always) to the data or content stored in it. Access cryptography applies the roads and networks for accessing a system.
  - Data: illustrated as the locks on the front door of a building. There may be several kinds of locks inside the building as well.
- **Discussion Topic: A Battle for Time?** [Presentation Slide 79] Explain to students that every cryptographic algorithm can be cracked, it is simply a matter of time. Most cryptographic algorithms and implementations are intentionally public knowledge to encourage the community to constantly improve security to keep pace with increasing computer speed.
- **Discussion Topic: Password Complexity?** [Presentation Slide 80] Explain to students that encryption methods and their complexity are built to increase the time it takes to decrypt them. Common password rules (weak, medium, strong) are built on the same logic and use the number of possible combinations needed to illustrate strength. Using the examples "icodes," "iluvcode," and "LuvCode3!" to illustrate how a password can become more complex, discuss with students that it takes 0.077228944 seconds, 52 hours, and 3 days to crack each of the example passwords. Explain to students that simply adding numbers or characters to a password does not make it strong. For example, assembling four random words like "correcthorsebatterystaple," which is made of all lowercase letters, no numbers or symbols, and complete words, would take a browser approximately one quintillion years to crack.
- **Discussion Topic: Encryption and Decryption** [Presentation Slides 81–84] Explain to students that an encryption algorithm uses a secret key to turn plain text (unencrypted data) into cipher text (encrypted data). Use the Caesar cipher example to illustrate how a basic secret key might work. In this example, the "key" is requiring that all letters be rotated to the right four letters, resulting in an encryption method that would make all "A" characters into "D" characters, "B" characters become "E" characters, etc. To decrypt the content of the cipher text you would need to understand the secret key and in the example it would be to reverse the rotation back four characters.

Teaching Tip

- The terms cryptography and encryption are used throughout Lesson Plan 4, Here's a quick reference to what these terms mean and how they're related to one another.
  - Cryptography: Securing a network's or program's most valuable assets.
  - Encryption: Maintains data confidentiality and requires the use of a secret key to return to plain text.
- It's helpful to equate cryptogtaphy to a general category like "colors" and encryption to a specific color like the color "blue."

Teaching Tip

To illustrate how cryptography and time are connected, use the website **https://howsecureismypassword.net** and have students type in their own personal passwords to see who in the class has the most secure password.

Discussion Topic: Hashing
[Presentation Slides 84–86] Explain to students that hashing is a one-way encryption method that takes plain text, such as passwords, and interprets them as a fixed digit code for validating later. Hashing can be thought of as a digital fingerprint or signature that is interpreted every time it is used. Platforms like Google and Facebook do not store a user's actual password, they store the hash key, which is interpreted and validated every time a user inputs his or her password. These systems only check to ensure that what the user typed in now is converted to the same hash code as what the user typed in before.

Explain to students that collision is when two different passwords are translated to the same fixed digit hash. To illustrate this concept, explain to students that because of an error in the hash algorithm the passwords "password123" and "iluvcode" could technically be stored in the database as the same hash, meaning you could use either to access your account.

▶ **Student Activity: Encryption Workshop**
[Presentation Slides 87–88]

Student Activity Objective: Students will work to plan and code a Caesar cipher (beginner and Intermediate) or a Vigenère cipher (advanced).

Instructions: Have students work as individuals or pairs to complete the Encryption Coding Challenge using the appropriate activity coding level. This activity should be introduced after the Lesson 4 classroom instruction and discussions have been completed.

Student Activity Tips

- Once students have completed the activity pair them with students from a similar coding level and challenge them to try and crack one another's ciphers.
- Students can work individually or in pairs on this activity. When they get stuck on a bug or an issue, have them review their progress with a student from the same coding level as them.
- Remind students that search engines and documentation can be powerful tools when repairing or debugging code.
- For best results, require students to plan their solution (using pencil and paper) or document their dependencies and issues before beginning to code. Activity levels can be scaled up or down depending on a student's ability.

Resources and Inspiration

Creating apps isn't easy, but there are many resources available to help. The links below are additional useful tools and references to guide and inspire you in teaching your students the path to app development. This document can also be downloaded at **scholastic.com/samsungacademy/resources.aspx**.

TECH NEWS AND STORIES

- www.techcrunch.com
- www.mashable.com
- www.engadget.com
- bits.blogs.nytimes.com
- open.blogs.nytimes.com
- online.wsj.news/technology
- www.readwrite.com
- www.smashingmagazine.com
- www.thenextweb.com
- www.theverge.com
- www.businessinsider.com/sai/mobile
- www.venturebeat.com/category/mobile

INNOVATION INSPIRATION

- www.samsung.com/us/business
- www.ted.com
- www.fastcompany.com
- www.psfk.com

DESIGN INSPIRATION

- android.inspired-ui.com
- www.pttrns.com
- www.lovelyui.com/tagged/android
- www.androidpatterns.com
- www.androidux.com
- www.androiduiux.com
- www.behance.net
- www.dribbble.com
- www.bittbox.com
- www.visual.ly
- www.uxmag.com
- www.thisiscolossal.com
- www.design.org
- www.ffffound.com
- www.hackdesign.org

WIREFRAMING AND PROTOTYPING

Keynote and PowerPoint can be great tools for creating simple wireframes and prototypes. And you probably already have access to one of them! See more on how to use Keynote as a prototyping tool at:

- www.keynotopia.com/guides
- www.balsamiq.com
- www.omnigroup.com/omnigraffle
- www.proto.io
- www.fluidui.com
- www.uxpin.com
- www.cacoo.com

PROGRAMMING AND DEVELOPMENT

- www.code.org
- www.codecademy.com
- www.khanacademy.org/cs
- www.teamtreehouse.com
- developer.samsung.com
- developer.android.com/index.html

Key Terms

Advertising (Paid Media): This form of marketing creates exposure for your application and is paid or purchased advertising or promotion. It can take place through various mediums (television, print, Internet, etc.).

Analytics: Developers set up analytics to collect data on various aspects of app usage. This information allows them to analyze the app to evaluate where the app is successful, and how the app can be improved in future updates.

Animation: Giving movement to digital objects.

API (Application Programming Interface): A system of routines, protocols, and tools used for building software applications.

App Inventor (MIT): A block-based programming tool that allows anyone to start programming and building fully functional apps for Android devices.

Audience Summary: Broad description of intended or assumed audience, and the kinds of people using applications.

Beta Test: A trial of software in the last phases of its development, conducted by someone not associated with its development.

Button: Allows a user to perform a specific action. Examples are tapping a button to travel to a different screen, or tapping a "submit" button to send information through the app.

Competitor/Competitive Analysis: Review of applications in the market with similar concepts, functionality, and capabilities. Includes competitor strengths/weaknesses, and strategies used by competitors.

Concept or Creative Brief: A document used by creative professionals and agencies to develop a broad creative concept and a time line for application development. The creative brief also allows for internal approval and is required before work can commence.

Connected Services: Offers developers the ability to access resources that may otherwise be too difficult or expensive to build (features, data, APIs or SDKs, etc.).

Cryptography: Securing a network's or program's most valuable assets.

Debugging: Process of finding and removing bugs, errors, and abnormalities from code that would otherwise cause unexpected behaviors in software

Developer: designs and writes software/computer programs to meet specific requirements

Documentation: The information that describes the product to its users.

Elevator Pitch or Elevator Speech: A short summary used to quickly and simply define a person, product, service, or organization, and its value, in the time span of an elevator ride, or approximately 30 seconds to two minutes.

Encryption: Maintains data confidentiality and requires the use of a secret key to return to plain text.

Friction: App developers aim to reduce friction by reducing the number of steps it takes a user to accomplish a task within the app.

Gestures: A hand movement used to control software on a mobile device.

Hashing: Validates the integrity of content by detecting all obvious changes to the hash output.

Home Screen: This is the first screen that appears when the app has loaded. The screen can be a log-in screen or an overview of the app contents such as navigation buttons and icons.

Icon: Often the symbol on which users click. Represents information and action, and helps users understand an app and what it can accomplish.

Iterative Development Cycle: The process of coding in repeated cycles of designing, developing, and testing. In this way, programmers create new versions of software.

Language or Code: How the developers program the app. There are multiple options, all of which allow different functions and experiences.

Market: For the purposes of this program, the term refers to the app market, and where apps enter and exist for consumers to download and use.

Minimum Viable Product (MVP): A product with just those core features that allow the product to be deployed and no more.

Mobile: Relating to cell phones, tablets, and other portable, wireless, handheld devices.

Key Terms
(Continued)

Mobile Application (App): Software that runs on a mobile device such as a smartphone and other mobile devices, and allows the device to perform specific tasks that are typically restrictive to computers.

Motion Animations: Animations that react to device hardware such as GPS, gyroscope, compass, etc.

Native App: An app written for a specific hardware platform. It will always run faster than a web app because there is no translation processing taking place.

Navigation: How a user travels through the app. This usually remains in a fixed spot within the user interface as buttons, and allows users to tap and go from screen to screen.

Nested Interaction: When a user interacts with an app via submenus. In other words, the user accesses a menu from a more general menu.

Pair Programming: Two programmers working side-by-side at a single computer. One person, the programmer or driver, writes code, while the other person, the navigator, guides and reviews each part of the code.

Physics Engines: Allows for dynamic interaction, to animate objects with physical properties, and within unique environments such as games and Sprite animations.

Platform (Hardware): Various electronic devices such as a notebook computer, mobile phone, personal digital assistant, music or video player, or handheld tablet.

Platform (Operating System Software): An application can be released on various platforms, such as an Android platform, BlackBerry, or Windows. The more platforms an app is released on, the more exposure and potential downloads an app will have.

Primary Navigation: Drives access to the app's core features.

Proof of Concept: A prototype of an app, meant to serve as a limited test of a concept or idea.

Purpose & Vision Summary: Answering basic goal, audience, and intent questions, this summary serves as the basic scaffolding on which an app and its solutions will be built. The summary is used throughout the process to define pathways.

Secondary Navigation: Additional navigation points.

Software Development Kit (SDK or "devkit"): A set of software development tools that allows for the creation of applications for a certain software package, software framework, hardware platform, computer system, video game console, operating system, or similar development platform.

Software Engineer: Responsible for the complete cycle of new or updated software products. Also designs, develops, and installs software solutions.

Splash Screen: The image that appears as the app is loading.

Sprite Animations: A two-dimensional image that is used within a larger scene to add motion to static objects.

Target Audience: A group of people in a desired market that a product or message is aimed toward.

Toggle: A key or command that switches between two electronic options.

Tone: How the design makes the user feel about the app.

Toolkit: Software included in an SDK that is designed to perform a specific function, especially to connect or control hardware or out-of-the-box functions.

User Experience (UX): The way a user interacts with the app, how he or she feels about the app, how easy it is to use the app, and how the app runs and functions.

User Interface (UI): How a user interacts with the visual experience of an app, which permits the user to navigate and use the application.

User Journey (UJ): Labeled flow diagram of steps that users take to complete tasks within an app.

User Profile: Creation of types (characters) with assigned attributes that represent target market, and can be referenced throughout the process. App features and tone are built to service the user profile, and assist in making future decisions.

Wireframe: A visual tool depicting labeled screens of the mobile device. Wireframes illustrate the functions, experience, and interface of the app.

Version 1.0: The first release of a software program. The software may not be entirely reliable at this point, and the developer may intend to add more features in the future.