



Introducing students to computer science through programming is a key 21st century skill and one that can open up opportunities for future careers. Incorporating computer programming into your learning setting can give all students access to a core skill and inspiring content. We recommend these five approaches to computer programming in the classroom.

## 1 Integrate programming with core subjects and real-world problems.

- Give all students access to learning programming through a multidisciplinary approach.
- Create context and relevance for programming by connecting it to core content.

Have students:

- Learn about sea-level rise impact through creation/manipulation of a computer simulation.
- Create a program to show the actions of a character in a favorite book.
- Create a computer game to simulate the effects of a dam on the environment.



## 2 Develop problem-solving skills through **computational thinking (CT)**.

- Intentionally use CT vocabulary and map class activities to CT concepts (abstraction, algorithms, decomposition, and pattern recognition).
- Use both *plugged* (on computer or device) and *unplugged* activities to develop CT skills.
- Unplugged activities are an opportunity to practice using CT skills to problem-solve without a computer.

Have students:

- Identify **patterns** using data gathered about sea-level rise.
- Use **decomposition** and **algorithms** to create a flowchart for a computer program.
- Develop a computer game in which they use **abstraction** to teach the main idea of a book.



To learn more about computational thinking and get ideas for how to use unplugged activities in the classroom, see our [Tech Tip on Computational Thinking](#).



## 3 Foster creativity through exploration.

- Teach students to be creators of technology, not simply consumers of technology.
- Provide students with opportunities to problem-solve through programming. Take a look at our [Sample Lesson Structure](#) for some examples of how to do this.

Have students:

- Explore and create their own programs.
- Give others feedback on their programs.
- Creatively enhance and customize existing programs.





4

Cultivate perseverance and leadership through **independent problem-solving**.

- Discuss and build strategies for coping and persevering. Rather than avoiding roadblocks and frustration, set expectations that students **will** experience this as part of their process.
- Build troubleshooting skills by encouraging students to find solutions. (see [CS Principle V](#)). Guide them with open-ended questions like “what have you tried/will you try?” or “what steps can you take to identify the problem?”

Have students:

- Use the rule “ask three then me” so students can support each other before they approach you.
- Do *unit testing* where they only test a section of code to see if it works as intended.
- Research the issue. When they find a solution, have them reference the source of inspiration as a comment in their program and share with other students.



5

Promote **collaboration** skills through peer communication.

- Highlight the ways in which software engineering and computer programming are highly collaborative.
- Reference the practices of software engineering teams and adapt them for the classroom.

Have students:

- Practice communicating and sharing ideas.
- Use pair programming, code reviews and stand-ups to collaborate, give and receive feedback.



 <b>Pair programming</b>	 <b>Code reviews</b>	 <b>Stand-ups</b>
<p>Two students share one device, each with a specific role.</p> <ul style="list-style-type: none"> <li>• “Driver” is concerned with the details of the code and they are the one writing the code (using keyboard, mouse, touchscreen, etc.)</li> <li>• “Navigator” asks questions and informs the “driver” of potential issues they may encounter. Drivers ask the navigator questions if they have concerns.</li> </ul> <p>After a set time (5-10 min) or block of code being completed, students switch roles.</p>	<p>Students exchange their code with one another and provide feedback on the program.</p> <ul style="list-style-type: none"> <li>• This can be a formal process with a feedback form, or an informal exchange when a student is struggling with a specific section in their program and needs input and ideas.</li> <li>• Code reviews can happen at any time in the process, and students do not need working programs to share and get feedback.</li> </ul>	<p>A quick way for the class to share their progress. Stand-ups should be short (less than 15 minutes) and only about the programming status.</p> <ul style="list-style-type: none"> <li>• This is an opportunity for other students to see what issues, tips, or ideas their peers have and how they are addressing them.</li> <li>• Do a stand-up as a class before everyone begins to work on their project, or have groups write up their progress and post it on a bulletin board in class or a digital collaboration space.</li> </ul>



**Learn More:** For more information on our approach to Computer Science, see [The Tech’s and SJSU’s Computer Science \(CS\) Principles](#).



## Collaboration in a Virtual Setting

Strategies for engaging students in computer programming via video conferencing (all students will need their own computer to participate):

**Pair programming** can be done through video conferencing with screen sharing.

- Model the process for students first by playing the role of “driver” and having students take turns “navigating” as you screen share.
- To have all students participate, pair them up and have them work in break-out rooms or smaller meetings.
- Once again, pair students and assign roles -- driver and navigator roles can still exist, with a few modifications.
  - The “driver” should share their screen and the “navigator” continues to ask questions and lets the driver know of potential issues.

**Virtual code reviews** are another option for distance learning.

- Have each student write what support they would like from their peers.
  - *Example:* “When my program gets to the first for loop, it.... But I want it to...”
- Pair students and have them exchange their code and support note. Each student will run and test the other’s program and provide feedback to their partner on how to address the issue and other ways to improve their code.
- This can be done asynchronously or in a virtual meeting.

## Terms and Definitions

**Code:** Set of written commands that a computer follows when executed.

**Computational Thinking:** Problem-solving process that can be broadly applied across content areas and everyday life. It includes: decomposition, algorithms, pattern recognition and abstraction

**Computer Programming:** Creation of computer programs to accomplish a task.

**Computer Science:** The study and practice of computer hardware and software, theory, applications and societal impact.

**Plugged activities:** Computer science activities that require the use of a computer or electronic device.

**Software Engineer:** A person who applies computer science to design and develop computer programs.

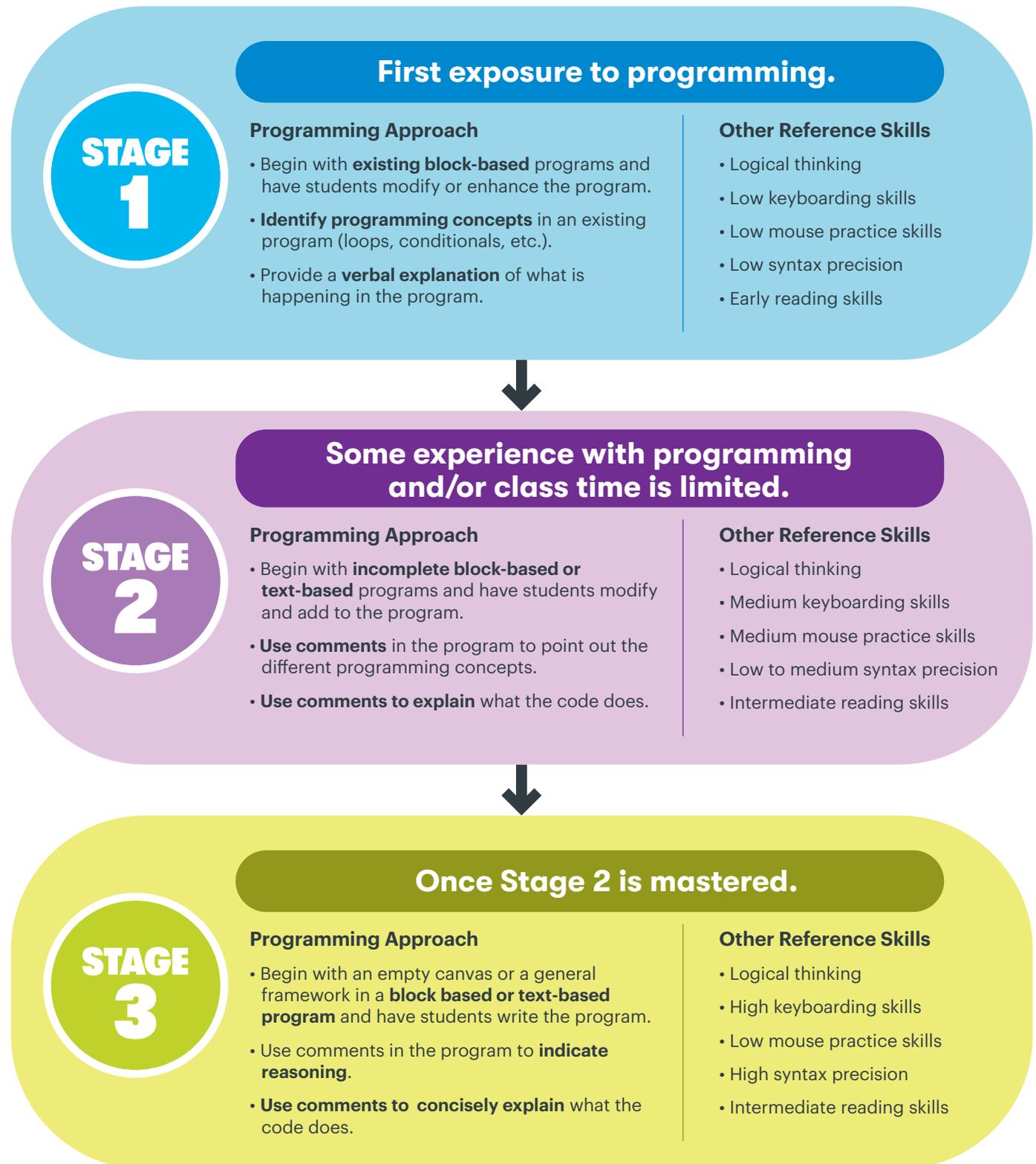
**Unit Testing:** Testing a section of code to see if it works as intended.

**Unplugged activities:** Computer science activities that do not require the use of a computer or electronic device. Can include a range of materials and settings: paper and pencil, games, puzzles, physical play etc.



## PROGRAMMING PROGRESSION

Students of all ages can learn how to program. Use the following progression as a guide for introducing students to computer programming. The starting level you choose will depend on grade level and programming familiarity of the students collectively. Keep in mind that you are likely to have a combination of programming levels in your classroom and students will progress at different paces.





## PREPARING FOR COMPUTER PROGRAMMING IN YOUR CLASSROOM

- Assign a specific device to students (sets of students). This will ensure that they become responsible for their own device.
- Have a key phrase that indicates you want students' attention on you and away from the screens.
  - One such phrase can be, "courtesy mode." This phrase will remind students to close their device, put the screen face down, or to close the screen at a 45 degree angle (out of their line of sight).

### Sample Programming Lesson Structure

Section	Time Frame	Tips
<b>Frame the Activity</b>	5-15 minutes	<ul style="list-style-type: none"> <li>• Use a compelling real-world problem to grab students' interest.</li> <li>• Connect to prior knowledge (integrate it with content!)</li> <li>• Draw connections between CT skill(s) students are going to use and how that is like the work of computer scientists.</li> <li>• Keep it short and sweet!</li> </ul> <p><i>Stage 2 example:</i> Watch a video of a butterfly emerging from its cocoon.</p>
<b>Unplugged Activity</b>	20-30 minutes	<ul style="list-style-type: none"> <li>• Before students get on computers, reinforce CT skills with an unplugged activity.</li> <li>• Focus on one main CT skill.</li> <li>• Use an activity that is directly related to the plugged design challenge.</li> </ul> <p><i>Stage 2 example:</i> Students create an <b>algorithm</b> (order) of the different stages in a butterfly's life cycle.</p>
<b>Plugged Design Challenge</b>	60-120+ minutes	<ul style="list-style-type: none"> <li>• Make the plugged design challenge an extension of the unplugged activity.</li> <li>• Start with an existing program that students can modify.</li> <li>• Have students test their code often!</li> <li>• Have students collaborate.</li> </ul> <p><i>Stage 2 example:</i> Students create a simulation in ScratchJr of the four stages of the butterfly's life cycle.</p>



### Adaptations

As with any other content area, differentiation is important. Some students may finish ahead of time or want to expand their programming knowledge. Offer these options as students finish their program.

- **Student choice:** Provide the opportunity for creative expression or enhancement of programs. This could be adding sound and narrating what is happening, changing colors of items, adding animation or anything that would enhance the overall computer program.
- **Debugging exercises:** Give students the same program or a repertoire of programs that have bugs and have them identify and suggest (multiple) fixes for the bugs.
- **Tech Leads:** Designate students to be Tech Leads for their peers. Have them provide support to students who need additional assistance.
  - It's important to let the Tech Leads know that they are not allowed to touch the computer of their peers. Rather, they need to guide them, ask questions and provide constructive feedback.
- **Prescriptive practice:** Build time into lessons for self-paced practice on prescriptive, step-by-step tools. Let students who need additional guided support review some of these modules when they encounter issues within an open environment.



### Educator Troubleshooting

Remember you are problem-solving too! Join a programming community so you have support when you get stuck. Many platforms have a large educator community where you can search for answers and post questions.



## TOOLS TO USE

The landscape for computer programming resources and educational tools is constantly changing. Here are a few tools and how they support our five approaches to programming.

Type of Platform	Grade	Programming Strengths				
		 integrated	 CT	 creative	 problem-solving	 collaborative
Block-based programming (open environment) <i>Ex: Scratch, MakeCode, Snap!</i>	K-12					
Block-based programming (prescriptive environment) <i>Ex: Code.org (CS Fundamentals), CSfirst.</i>	K-12					
Text programming (open environment) <i>Ex: Python, Java, C++.</i>	6-12					
Text programming (prescriptive environment) <i>Ex: CodeHS, Grasshopper.</i>	6-12					
Hardware* <i>Ex: Microbit, Arduino, Raspberry Pi, TI.</i>	4-12					
Robotics** <i>Ex: Sphero, Bee-Bot, Lego Mindstorm.</i>	K-12	 (some)		 (some)		

\*Hardware and a device will be needed to program.  
 \*\*Hardware and sometimes a device may be needed to program — do research to learn more.