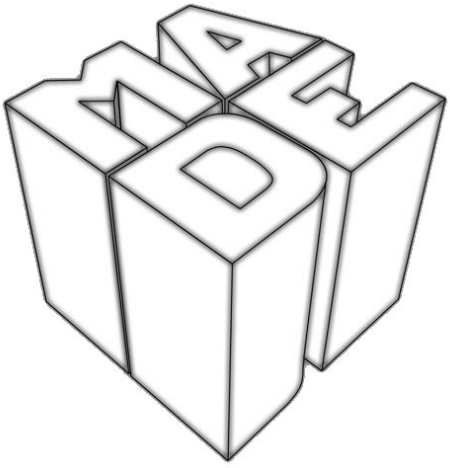
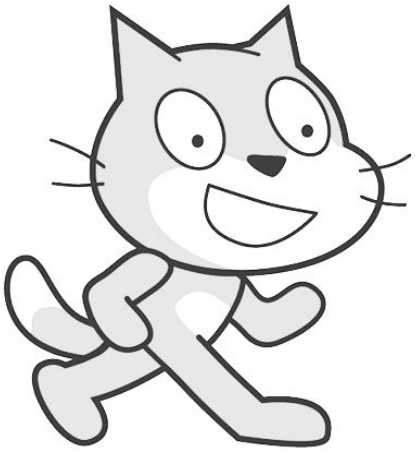
Scratch Class Handbook





The Museum of Art and Digital Entertainment
34000 Broadway, Oakland, CA 94611

The MADE is the only all-playable video game museum in the world. We were the first dedicated open to the public video game museum in the United States. Our collection houses over 5,300 playable games. The MADE is a 501c3 nonprofit dedicated to the preservation of video game history, and to educating the public on how video games are created. Our goal is to inspire the next generation of game developers.

Edited by Al Sweigart al@inventwithpython.com
Version 3, Last modified 2020/10/07

Version History

Version 3 - 2020/10/07 - Updates for Scratch 3 and the Scratch Project Ideas section.
Version 2 - 2020/02/26 - Added Phil Agre's tips section and advice on building a computer lab.
Version 1 - 2018/11/07 - Initial release.

About

This document contains information relevant to the Saturday Scratch Programming class at the Museum of Digital Art and Entertainment (the MADE), but it could be relevant for anyone teaching a programming class or after school coding club.

This document covers Scratch 3. This document can still be useful for those using Scratch 2 since it gives advice on teaching Scratch, rather than Scratch programming itself.

Setting Up the Computer Lab

Whether you have a laptop cart or students are bringing in their own computers, go through the following steps to set up the computer for Scratch programming.

- Download and install the offline Scratch Desktop editor from https://scratch.mit.edu/download/. The offline editor looks identical to the browser-based editor, but will continue working if you lose wifi, though you can still upload projects to a Scratch account. (On the other hand, using the browser-based will automatically save projects in case the browser crashes, while you must remember to manually save using the offline editor.) The older Scratch 2 and Scratch 1.4 offline editors are also available for download on this page, but you should use the latest version.
- Go to https://scratch.mit.edu and set up a Scratch account. These accounts are free and only require an email address (which is used only for password recovery.) Students can use their own Scratch accounts, but you should set up Scratch accounts beforehand for students to use. You can print and hand out small slips of paper with the Scratch URL, username, and password on them for students to use. (You can use the same email address when signing up for multiple accounts.)
- Alternatively, you can set up a single account for the class and share the password (however, this does mean any student can change the password and lock everyone else out of the account.) Change the password (by signing in, clicking your username in the upper right, then go to Account Settings, then click the Password tab) to something simple before the start of the class (i.e. "applejack") so that students can use it to upload their projects, and then change it back to a secret, strong password (i.e. "6Hq3CcVeiyf") at the end of class.

If you are building a computer lab from scratch, we've found that cheap $200 to $300 laptops are the best way to go, along with external USB mice. Creating a computer lab with Raspberry

Pi's requires so many accessories (casing, keyboard & mouse, monitor, power cables, etc) that there are no cost savings but lots of setup/teardown headache. Additionally, using desktop computers is fine but makes it difficult to rearrange the computer lab space or move it to different rooms as needed. As of Scratch 3, Scratch can run on tablets and phones, though tablets are recommended since they provide more screen real estate.

Pre-Class Checklist

Follow these steps before the start of each class:

- Open Scratch Desktop offline editor and **maximize the window** (students forget to maximize the window and continue to work in a small window).
- Mute each computer. (Sounds are distracting.)
- Make sure each computer has a mouse and mousepad. Students tend to prefer the touchpad (probably from experience using phones with touch screens), but the mouse is a much more effective tool for dragging code blocks in the editor.
- Have a pen & notepad to write down improvement class ideas as you have them. (If you wait until after the class, you'll forget them.) Use paper: if you write these down on your phone, it looks like you're goofing off on your phone to the students who will probably want to do the same.
- Download any images for the project to the desktop. It's too time-consuming to have students download images for projects themselves. You don't want students searching for images to use on Google Image Search since the adult filter is not always 100% effective. If possible, stick to the images in the Scratch sprite and backdrop library.
- If you are projecting from your laptop, it can be hard for students to see the blocks you're using. Click the zoom button (the magnifying-glass-with-plus-sign icon in the Code Area) a few times. This increases the block size and makes it easier to see from the back of the class. (Although a self-paced classes are better than lecture classes. See the "Class Style" section in this document.)
- If you are projecting from your laptop, make your mouse pointer easier to see on the screen by choosing larger mouse pointers. On Windows, go to the Control Panel or Mouse Settings, set mouse pointer scheme to "Windows Aero (extra large)" for better visibility.
- Physically walk to the back of the room to look at the projected screen to remind yourself how far away it looks for students.
- Make sure each computer's screen saver/auto log off is disabled. These tend to interrupt the student's work if they haven't moved the mouse in a few minutes.
- Draw the layout of seats in the classroom on paper so you can fill it in with their names as they introduce themselves. This can be a vital memory aid.

Start of Class Checklist

Follow these steps at the start of each class.

- Tell parents that they can sit with their kids and work along with them, or wait in the lobby.
- Make the effort to memorize names, and write them down. Have the students introduce themselves by saying their name and favorite video game. As an example, teachers should introduce themselves first. Draw the layout of seat on paper, and fill it in with the students' names as they introduce themselves so you can remember who is who. It's hard to get a student's attention if you don't know their name! Have your name written on the whiteboard in large letters where students can see it.
- Demo the completed project that they'll make so they can see what it is they're working towards.

Teaching Tips

Keep these tips in mind while teaching the class.

- KEEP YOUR HANDS OFF THE STUDENT'S KEYBOARD AND MOUSE. When directing them to type and click, point to the screen where they should click or the keyboard keys they should press. (Young adults might not have keyboard skills and need help locating keys.) Don't take over the computer and do the task for them. Pointing with your finger is slower, but doing the task for them means they won't get the experience of coding. They learn from doing, not watching you do it.
- Remember to have the students save their work every 15 minutes or so. Make an announcement. The filename format should be "<student name> - <project name>" so that there aren't multiple projects with the same name on the computer. (This doesn't apply if they're using the browser-based Scratch editor instead of the offline editor.)
- If you are typing on your computer, be sure to face towards the class. Don't talk at the front wall. If you are on one of the desktops, turn the monitor & keyboard around to face the class.
- If the class is loud and you need to get people's attention, a classic teacher trick is to say "if you can hear my voice, clap once. <You clap once too.> If you can hear my voice, clap twice. <You clap twice too.>" Try to only use this trick once per class.
- The point of this class is **not** so that the students learn to program but that 1) they think programming is cool and 2) they think programming is something they're capable of.
- **ENCOURAGE and PRAISE them.** Say "Good job", "Yeah, you got it", & "Nice drawing".
- Students have a hard time seeing the projector from the back. Make sure the editor zoom is at max zoom and stays that way.
- Keep in mind that stuff at the top of the projector screen is easy to see; stuff at the bottom is hard to see.
- Don't block the projector. Students should be able to always see you **and** the screen at all times.

- Uninstall the old version 1.4 and version 2 offline editors if you find it on a machine. These are the old version of Scratch and there's no reason to have it. Students may be confused if they launch the wrong version of Scratch.

Class Styles

Originally, Scratch classes at the MADE were 90 minutes long and an instructor on a projector would guide students through a small project. By the end of the class, they would have a small video game or other program that they made. The downside of this format is that the class can only move as fast as the slowest student, leaving other students open to distraction. If they had any bugs in their code, it could take a while to discover them.

As a better alternative, we created web-based handouts with self-paced instructions for each step of the project. Prototypes of these are available at https://inventwithscratch.com/tutorials/. As students worked through the steps, instructors could float and offer help as needed. The downside is that it takes a while to create these handouts.

The best format for these handouts were short, looped, animated gifs demonstrating each of these steps. These animated gifs were made with the LICEcap program for Windows, and placed into small web pages. If you don't know how to create web pages, they could also be placed into PowerPoint slides. Unlike static images, they show the exact process of adding blocks or clicking in the editor. Animated gifs avoid the problem of videos in that students don't have to constantly pause and rewind to review instructions. Animated gifs are also better than written instructions: "A picture is worth a thousand words". However, being animated, these cannot be printed out on paper.

We'd eventually like to create slide decks of these animated gifs. Rough drafts of them can be found at https://inventwithscratch.com/tutorials/.

If you are interested in creating more of these projects to distribute to other instructors, please contact Al Sweigart at al@inventwithpython.com.

Project Tips

When choosing a programming project for the class, keep in mind the following tips.

- **Keep in mind your project will take longer to make than you think.** The point of this class is to make Scratch programming seem cool and something they are capable of doing; they don't actually have to learn coding or how to make complex projects.
- **Video tutorials of various Scratch projects can be found at https://inventwithscratch.com.** You can also find Al Sweigart's Scratch projects at https://scratch.mit.edu/users/AlSweigart/.

- **The book, "Scratch Programming Playground" has several projects and is free to download from https://inventwithscratch.com.**
- Avoid games that require a camera view (i.e. camerax and cameray variables). This is a complex programming concept for students.
- Programs should have at most three or four variables. Requiring more than this may be a sign that your program is too complicated.
- If you have a sprite named "foo", don't use "foo" for variable or broadcast message names.
- Professional software developers might design their code to be "generalized" and "elegant", but this means it's probably too hard or abstract to understand. It's okay to have copy/pasted code because it'll be easier to understand.
- Code that uses indirection/abstract concepts (indexes to lists, magic numbers, etc) is hard to understand.
- Be a part of the vowel generation: Use variable names like "string compare", not "strcmp".
- Don't use literal guns in the game. Alternatives: bow & arrow, "energy balls", lightning bolts, large cannons or catapults that shoot bowling balls.
- If you have shooting targets, make sure they aren't living things. Have the students shoot inanimate objects (asteroids, targets, apples, balloons, stars, sprites from the "Things" category in the sprite library) instead.
- Stick to the images in the Sprite Library and Backdrop Library. If the students have to draw their own sprite, make sure it is a simple picture. Drawing sprites eats up a lot of time, the size of the sprite can be too big/small, or the costume center is off, etc. **Students drawing sprites always ends up taking up most of the time in the class.**
- Avoid changing the costume center if necessary. (In fact, avoid needing the paint editor as much as possible.)
- **Never** have the students find their own images on Google Image Search, even if the "filter adult images" option is enabled on the browser. The filter isn't 100% effective. Even if this isn't an issue, students will spend a lot of time searching and choosing images.
- If using non-library images and sounds, don't have the student download the files themselves. Download it to the desktop before the start of the class. Ideally, stick to the images and sounds in Scratch's library.
- For project ideas, use mini-games that appear in real video games. Or take an item from a Zelda game like the bow, hookshot, lantern, or boomerang and make a game that uses just that item.
- Think about what the "Minimum Viable Product" would be for your project: Have the students design a basic game that works, and then let them add additional features later. This way, even if the class runs behind, they'll still end with a playable game instead of an unfinished project.

Common Coding Mistakes

These are common mistakes that students make when coding for themselves or even copying code from someone else.

- Make sure “For all sprites” or “For this sprite only” is properly selected. On the Stage, “For this sprite only” variables will have the sprite name in front of the variable name.
- If the “goto x y” code seems to make the sprite move somewhere else, the costume center might be off.
- Students use “set” instead of “change” for the “set/change x” or “set/change variable” or “set/change color effect” blocks.
- Students mix up the <, =, and > blocks.
- Students will add code to the wrong sprite. Check which sprite is selected when looking at their code.
- Students use “broadcast” instead of “broadcast and wait”.
- When using <, =, >, make sure whitespace isn’t accidentally entered in the text fields.

Post Class Checklist

Follow these steps at the end of each class.

- Go over the bullet list of what different features they made. Say something about how “it doesn’t take a lot to make a game that is fun”.
- Thank students by name for coming to the class.
- Hand out post-class cards (preferably to parents if they’re there.) These are cards that tell the students how to find the Scratch website and show their games to other people. (This is an important part; letting students show off their work makes them more enthusiastic to continue it.)
- If you need help getting students out of the class, ask them if they’ve saved their project, then tell them they can shut down the Scratch editor and close the laptop lid. This will keep them from lagging behind.
- Record the number of students, their names, the instructors, the project, and other details in a class spreadsheet. This information is useful for demonstrating the efficacy of your class to fund raisers and donors.

Scratch Project Ideas

Here’s a list of games and projects that are simple enough to be created in a 90 minute class. You can find many of them on Al Sweigart’s Scratch profile at https://scratch.mit.edu/users/AlSweigart/projects/:

- Pong / Brick Breaker - Move a “paddle” at the bottom of the screen to bounce a ball against objects at the top of the screen.

- Ikaruga (color switch/matching game) - Switch the color of a character between red, green, and blue, collecting apples of the same color, but don't touch the apples of different colors.
- Two-Key Race - Two players must alternate between pressing the A and S (player one) or K and L (player two) keys to move forward. Pressing the wrong key moves the player backwards.
- Maze - Move a player through a maze. Maze image files can be downloaded from http://www.mazegenerator.net/.
- Tron - Players are constantly moving forward and can only control their direction but not speed. The player sprites draw a line behind them (using the Pen blocks) and the first player to crash into a line loses.
- Dodge Game - A player moves at a set speed and must dodge objects as they fall from the top of the screen.

Phil Agre's "How to Help Someone Use a Computer"

Phil Agre has a collection of excellent tips at https://pages.gseis.ucla.edu/faculty/agre/how-to-help.html. The document is repeated here.

Computer people are fine human beings, but they do a lot of harm in the ways they "help" other people with their computer problems. Now that we're trying to get everyone online, I thought it might be helpful to write down everything I've been taught about helping people use computers.

First you have to tell yourself some things:

- Nobody is born knowing this stuff.
- You've forgotten what it's like to be a beginner.
- If it's not obvious to them, it's not obvious.
- A computer is a means to an end. The person you're helping probably cares mostly about the end. This is reasonable.
- Their knowledge of the computer is grounded in what they can do and see -- "when I do this, it does that". They need to develop a deeper understanding, but this can only happen slowly -- and not through abstract theory but through the real, concrete situations they encounter in their work.
- Beginners face a language problem: they can't ask questions because they don't know what the words mean, they can't know what the words mean until they can successfully use the system, and they can't successfully use the system because they can't ask questions.
- You are the voice of authority. Your words can wound.
- Computers often present their users with textual messages, but the users often don't read them.
- By the time they ask you for help, they've probably tried several things. As a result, their computer might be in a strange state. This is natural.

- They might be afraid that you're going to blame them for the problem.
- The best way to learn is through apprenticeship -- that is, by doing some real task together with someone who has a different set of skills.
- Your primary goal is not to solve their problem. Your primary goal is to help them become one notch more capable of solving their problem on their own. So it's okay if they take notes.
- Most user interfaces are terrible. When people make mistakes it's usually the fault of the interface. You've forgotten how many ways you've learned to adapt to bad interfaces.
- Knowledge lives in communities, not individuals. A computer user who's part of a community of computer users will have an easier time than one who isn't.

Having convinced yourself of these things, you are more likely to follow some important rules:

- Don't take the keyboard. Let them do all the typing, even if it's slower that way, and even if you have to point them to every key they need to type. That's the only way they're going to learn from the interaction.
- Find out what they're really trying to do. Is there another way to go about it?
- Maybe they can't tell you what they've done or what happened. In this case you can ask them what they are trying to do and say, "Show me how you do that".
- Attend to the symbolism of the interaction. Try to squat down so your eyes are just below the level of theirs. When they're looking at the computer, look at the computer. When they're looking at you, look back at them.
- When they do something wrong, don't say "no" or "that's wrong". They'll often respond by doing something else that's wrong. Instead, just tell them what to do and why.
- Try not to ask yes-or-no questions. Nobody wants to look foolish, so their answer is likely to be a guess. "Did you attach to the file server?" will get you less information than "What did you do after you turned the computer on?".
- Explain your thinking. Don't make it mysterious. If something is true, show them how they can see it's true. When you don't know, say "I don't know". When you're guessing, say "let's try ... because ...". Resist the temptation to appear all-knowing. Help them learn to think the problem through.
- Be aware of how abstract your language is. "Get into the editor" is abstract and "press this key" is concrete. Don't say anything unless you intend for them to understand it. Keep adjusting your language downward towards concrete units until they start to get it, then slowly adjust back up towards greater abstraction so long as they're following you. When formulating a take-home lesson ("when it does this and that, you should try such-and-such"), check once again that you're using language of the right degree of abstraction for this user right now.
- Tell them to really read the messages, such as errors, that the computer generates.
- Whenever they start to blame themselves, respond by blaming the computer. Then keep on blaming the computer, no matter how many times it takes, in a calm, authoritative tone of voice. If you need to show off, show off your ability to criticize bad design. When

they get nailed by a false assumption about the computer's behavior, tell them their assumption was reasonable. Tell *yourself* that it was reasonable.

- Take a long-term view. Who do users in this community get help from? If you focus on building that person's skills, the skills will diffuse to everyone else.
- Never do something for someone that they are capable of doing for themselves.
- Don't say "it's in the manual". (You knew that.)