Problem Set 0: Scratch

This is CS50. Harvard University. Fall 2015.

Table of Contents

Objectives

- Understand how information can be represented digitally.
- Leverage some fundamental programming constructs.
- Design your own animation, game, or interactive art.
- Impress your friends.

Academic Honesty

This course's philosophy on academic honesty is best stated as "be reasonable." The course recognizes that interactions with classmates and others can facilitate mastery of the course's material. However, there remains a line between enlisting the help of another and submitting the work of another. This policy characterizes both sides of that line.

The essence of all work that you submit to this course must be your own. Collaboration on problem sets is not permitted except to the extent that you may ask classmates and others for help so long as that help does not reduce to another doing your work for you. Generally speaking, when asking for help, you may show your code to others, but you may not view theirs, so long as you and they respect this policy's other constraints. Collaboration on

quizzes is not permitted at all. Collaboration on the course's final project is permitted to the extent prescribed by its specification.

Below are rules of thumb that (inexhaustively) characterize acts that the course considers reasonable and not reasonable. If in doubt as to whether some act is reasonable, do not commit it until you solicit and receive approval in writing from the course's heads. Acts considered not reasonable by the course are handled harshly. If the course refers some matter for disciplinary action and the outcome is punitive, the course reserves the right to impose local sanctions on top of that outcome that may include an unsatisfactory or failing grade for work submitted or for the course itself.

If you commit some act that is not reasonable but bring it to the attention of the course's heads within 72 hours, the course may impose local sanctions that may include an unsatisfactory or failing grade for work submitted, but the course will not refer the matter for further disciplinary action except in cases of repeated acts.

Reasonable

- Communicating with classmates about problem sets' problems in English (or some other spoken language).
- Discussing the course's material with others in order to understand it better.
- Helping a classmate identify a bug in his or her code at office hours, elsewhere, or even online, as by viewing, compiling, or running his or her code, even on your own computer.
- Incorporating snippets of code that you find online or elsewhere into your own code, provided that those snippets are not themselves solutions to assigned problems and that you cite the snippets' origins.
- Reviewing past semesters' quizzes and solutions thereto.
- Sending or showing code that you've written to someone, possibly a classmate, so that he or she might help you identify and fix a bug.
- Sharing snippets of your own code online so that others might help you identify and fix a bug.
- Turning to the web or elsewhere for instruction beyond the course's own, for references, and for solutions to technical difficulties, but not for outright solutions to problem set's problems or your own final project.

- Whiteboarding solutions to problem sets with others using diagrams or pseudocode but not actual code.
- Working with (and even paying) a tutor to help you with the course, provided the tutor does not do your work for you.

Not Reasonable

- Accessing a solution to some problem prior to (re-)submitting your own.
- Asking a classmate to see his or her solution to a problem set's problem before (re-)submitting your own.
- Decompiling, deobfuscating, or disassembling the staff's solutions to problem sets.
- Failing to cite (as with comments) the origins of code or techniques that you discover outside of the course's own lessons and integrate into your own work, even while respecting this policy's other constraints.
- Giving or showing to a classmate a solution to a problem set's problem when it is he or she, and not you, who is struggling to solve it.
- Looking at another individual's work during a quiz.
- Paying or offering to pay an individual for work that you may submit as (part of) your own.
- Providing or making available solutions to problem sets to individuals who might take this course in the future.
- Searching for, soliciting, or viewing a quiz's questions or answers prior to taking the quiz.
- Searching for or soliciting outright solutions to problem sets online or elsewhere.
- Splitting a problem set's workload with another individual and combining your work.
- Submitting (after possibly modifying) the work of another individual beyond allowed snippets.
- Submitting the same or similar work to this course that you have submitted or will submit to another.
- Submitting work to this course that you intend to use outside of the course (e.g., for a job) without prior approval from the course's heads.
- Using resources during a quiz beyond those explicitly allowed in the quiz's instructions.

- Viewing another's solution to a problem set's problem and basing your own solution on it.

Assessment

Your work on this problem set will be evaluated along two axes primarily.

Scope
: To what extent does your code implement the features required by our specification?

Correctness
: To what extent is your code free of bugs?

All students, whether taking the course SAT/UNS (at Harvard), Credit/D/Fail (at Yale), or for a letter grade (at Harvard or Yale), must ordinarily submit this and all other problem sets to be eligible for a satisfactory grade unless granted an exception in writing by the course's heads.

1001000 1001001

First curl up with Nate's short on binary, if not too familiar:

https://www.youtube.com/watch?v=hacBFrgtQjQ
And next with Nate's short on ASCII:

https://www.youtube.com/watch?v=UPlR4eMMCmI
Then tinker with Binary Bulbs, below, by Michael Ge '18. Specifically, turn on Game Mode a few times and challenge yourself with these inputs, one game at a time:

- 50
- 127
- 42
- 256

If you'd like, turn the bulbs off to see the underlying bits that they represent. Better yet, turn the labels off to challenge yourself all the more.

<iframe></iframe>

Alright, consider these questions rhetorical for now, but odds are they'll come up again! Not to worry if the answers aren't obvious at first. They're meant to induce a bit of thought! Week 0's first lecture and Nate's videos should provide you with the building blocks (daresay inputs!) with which to solve these problems.

- How do you represent the (decimal) integer 50 in binary?
- How many bits must be "flipped" (i.e., changed from 0 to 1 or from 1 to 0) in order to capitalize a lowercase `a` that's represented in ASCII?
- How do you represent the (decimal) integer 50 in, oh, "hexadecimal," otherwise known as "base-16"? Know that decimal is considered "base-10" (since it employs 10 digits, 0 through 9), and binary is considered "base-2" (since it employs 2 digits, 0 and 1). Infer from those base systems how to represent base-16! (We'll see base-16 again in the context of graphics and web programming.)

Itching to Program?

Head to https://scratch.mit.edu/ and sign up for an account on MIT's website by clicking **Join Scratch** atop the page. Any username (that's available) is fine, but take care to remember it and your choice of password.

Then head to https://scratch.mit.edu/help/ and take note of the resources available to you before you dive into Scratch itself. In particular, you might want to skim the Getting Started Guide[1].

Next try your hand at *Pikachu's Pastry Catch* by Gabe Walker! Click the green flagand then, per Gabe's instructions, hit your keyboard's space bar, at which point the game will begin! Feel free to procrastinate a bit.

<iframe></iframe>

If curious, Gabe's source code can be seen at http://scratch.mit.edu/projects/26329354/. (You can also full-screen the game at that same URL, as full-screening the embedded game here might not work.)

Next, be sure you know what's recyclable and compostable these days by trying out this remix of *Oscartime* by Jordan Hayashi!

[1] https://cdn.scratch.mit.edu/scratchr2/static/__95f8025b5d5663c8eca07b96a66ef8d6__/pdfs/help/Getting-Started-Guide-Scratch2.pdf

<iframe></iframe>
Jordan's source code can be found at https://scratch.mit.edu/projects/71161586/. (You can also full-screen that game at that same URL.)

If you've no experience (or comfort) whatsoever with programming, rest assured that Gabe's and Carlos's projects are more complex than what we expect for this first problem set. (Click **See inside** in Scratch's top-right corner to look at each project's underlying "implementation details.") But they do reveal what you can do with Scratch.

In fact, for a gentler introduction to Scratch (and programming more generally), you might want to review some of the examples that we looked at in Week 0's second lecture and take a look at a few more, the "source code" for which can be found at http://scratch.mit.edu/studios/1500610/. Allow me to take you on a tour, though feel free to forge ahead on your own if you'd prefer:

https://www.youtube.com/watch?v=tveoFN0NHE0&list=PLhQjrBD2T383nc2LUdF5XWbyrsqiYy4nq
And you might also want to watch Allison's short on Scratch:

https://www.youtube.com/watch?v=52JoFF4HMA4
Feel free to download the source code for a few more projects from http://scratch.mit.edu/explore/projects/all/ or elsewhere. For each program, run it to see how it works overall and then look over its scripts to understand how it works underneath the hood. Feel free to make changes to scripts and observe the effects. Once you can say to yourself, "Okay, I think I get this," you're ready to proceed.

Now it's time to choose your own adventure! Your mission is, quite simply, to have fun with Scratch and implement a project of your choice (be it an animation, a game, interactive art, or anything else), subject only to the following requirements.

- Your project must have at least two sprites, at least one of which must resemble something other than a cat.
- Your project must have at least three scripts total (i.e., not necessarily three per sprite).
- Your project must use at least one condition, one loop, and one variable.
- Your project must use at least one sound.
- Your project should be more complex than most of those demonstrated in lecture (many of which, though instructive, were quite short) but it can be less complex than, say,

Pikachu's Pastry Catch and *Ivy's Hardest Game*. As such, your project should probably use a few dozen puzzle pieces overall.

Feel free to peruse additional projects online for inspiration, but your own project should not be terribly similar to any of them. Try to think of an idea on your own, and then set out to implement it. But don't try to implement the entirety of your project all at once: pluck off one piece at a time. Gabe, for instance, probably implemented just one pastry first, before he moved onto the game's other sprites. And Carlos probably implemented Yale before he moved on to implementing MIT. In other words, take baby steps: write a bit of code (i.e., drag and drop a few puzzle pieces), test, write a bit more, test, and so forth.

If, along the way, you find it too difficult to implement some feature, try not to fret; alter your design or work around the problem. If you set out to implement an idea that you find fun, you should not find it hard to satisfy this problem set's requirements.

Alright, off you go. Make us proud!

Not quite sure how to begin? Feeling a bit overwhelmed? Not to worry. Join Zamyla for a walkthrough of this problem set, if you'd like more of a tour:

https://www.youtube.com/watch?v=697pD31GCZg
Incidentally, if you don't have the best Internet access, you're welcome to download Scratch's "offline editor" at https://scratch.mit.edu/scratch2download/. But when done with your project offline, be sure to upload it to your account at http://scratch.mit.edu/ via **File > Share to website** in the offline editor.

Once finished with your project, click **See project page** in Scratch's top-right corner. Ensure your project has a title (in Scratch's top-left corner), some instructions (in Scratch's top-right corner), and some notes and/or credits (in Scratch's bottom-right corner). Then click **Share** in Scratch's top-right corner so that others (e.g., your TF!) can see your project. Finally, take note of the URL in your browser's address bar. That's your project's URL on MIT's website, and you'll need to know it later.

Oh, and if you'd like to exhibit your project in Fall 2015's studio, head to https://scratch.mit.edu/studios/1493562/, then click **Add projects**, and paste in your own project's URL.

Incidentally, if curious to learn more about the design of Scratch itself, you might like this segment with our friends from MIT's Media Lab:

https://www.youtube.com/watch?v=iLSBUKs4AYU

How to Submit

To submit this problem set, visit https://forms.cs50.net/2015/fall/psets/0. You'll find that a few questions await. Be extra-sure that your answers are correct, particularly your project's URL on MIT's website, lest we overlook your submission!

CS50 collects quite a bit of data for research purposes. Please forgive the length of this first problem set's form! In total, the form will probably take 15 or more minutes to fill out. Not to worry if you finish shortly after the problem set's deadline, so long as you start before the deadline. Future problem sets' forms will be much quicker!

This was Problem Set 0.