Problem 1-0: Scratch

This is CS50. Harvard University. Fall 2014.

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 problems is not permitted (unless explicitly stated otherwise) 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 or writing to others, but you may not view theirs, so long as you and they respect this policy's other constraints. Collaboration on quizzes and tests is not permitted at all. Collaboration on the 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 your instructor. If a violation of this policy is suspected and confirmed, your instructor reserves the right to impose local sanctions on top of any disciplinary outcome that may include an unsatisfactory or failing grade for work submitted or for the course itself.

Reasonable

- Communicating with classmates about 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, such 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 years' quizzes, tests, 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 solutions to problems online so that others might help you identify and fix a bug or other issue.
- 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 problems or your own final project.
- Whiteboarding solutions to problems 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 before (re-)submitting your own.
- Decompiling, deobfuscating, or disassembling the staff's solutions to problems.
- Failing to cite (as with comments) the origins of code, writing, 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 when it is he or she, and not you, who is struggling to solve it.
- Looking at another individual's work during a quiz or test.
- Paying or offering to pay an individual for work that you may submit as (part of) your own.
- Providing or making available solutions to problems 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 problems online or elsewhere.
- Splitting a problem's workload with another individual and combining your work (unless explicitly authorized by the problem itself).
- 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.
- Using resources during a quiz beyond those explicitly allowed in the quiz's instructions.
- Viewing another's solution to a problem and basing your own solution on it.

Assessment

Your work on this problem 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?

To obtain a passing grade in this course, all students must ordinarily submit all assigned problems unless granted an exception in writing by the instructor.

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

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! The material you’ve seen so far, along with 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 design later in the course.)

Incidentally, ever seen a *really* old computer? In the Harvard Science Center, there’s a big machine by the main stairwell. It’s the Harvard Mark I[1], one of the world’s earliest (electro-mechanical) computers, whose purpose was, well, to compute (albeit slowly by today’s standards)! You might enjoy this tour by Prof. Harry Lewis[2]:

https://www.youtube.com/watch?v=4ObouwCHk8w

[1] http://en.wikipedia.org/wiki/Harvard_Mark_I
[2] http://lewis.seas.harvard.edu/

Itching to Program?

Head to http://scratch.mit.edu/ and sign up for an account (if you are 13 or older) 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 http://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[3].

Next head to http://scratch.mit.edu/projects/26329354/ to see *Pikachu's Pastry Catch* by Gabe Walker. Click the blue square above the game's top-left corner if you'd like to full-screen the user interface (UI). Then click the green flag. Per Gabe's instructions, as soon as you hit your keyboard's space bar, the game will begin! Feel free to procrastinate a bit. And if you'd like to try out *Ivy's Hardest Game*, by Carlos Peña-Lobel, head to http://scratch.mit.edu/projects/26329347/.

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 problem. (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 you may have seen previously (and take a look at a few more), the "source code" for which can be found at http://scratch.mit.edu/studios/522341/. Allow David 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

[3] http://scratch.mit.edu/scratchr2/static/__1378420408__//pdfs/help/Getting-Started-Guide-Scratch2.pdf

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 (which, to be clear, must all be implemented in order to obtain full Scope points).

- 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 some basic examples, but it can be less complex than, say, *Pikachu’s Pastry Catch* and *Ivy’s Hardest Game*. As such, your project should probably use at least 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’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, 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 http://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 teacher!) 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 CS50 AP 1516's studio with students from around the country taking CS50 AP alongside you, head to https://scratch.mit.edu/studios/1493564/, 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
This was Problem 1-0, your first programming problem in CS50 AP!