

Getting a computer to work for you. Throughout this course I will be trying to convince you that statistical mechanics is a beautiful subject, one that explains how order emerges from the randomness of large collections of objects. One way to probe that order is to manufacture our own randomness, generating samples on a computer and analyzing them. Historically, learning statistical mechanics therefore meant simultaneously learning to write and run the programs that generate and analyze those samples.

The manner of forcing a computer to do your bidding has changed over time. Once upon a time you really needed to understand binary logic and to spell out sequences of operations on the ones and zeros stored in memory. Then people built compilers, programs that turn higher-level instructions into those ones and zeros, so that you could put words on a screen by typing “print” rather than an illegible string of bits. Languages accumulated, each letting people express more directly the essence of what they wanted the machine to do. Knowing how to use those languages has always been a different thing from understanding statistical mechanics, but it has typically been a necessary step toward the computational experiments that build understanding.

While scientists and engineers were building more convenient ways to talk to the ones and zeros, so were business people. They revolutionized how most lay people view a computer or a phone, to the point that you rarely think about its general-purpose computing ability at all. You think about Chrome, Outlook, Excel, Word, and TikTok. I have little doubt that you are fluent in those products. I find that many students are much less familiar with how to reach the general-purpose power sitting inside their machine.

Some of the problem sets will ask you to reach that power, so in past years I wrote a gentle Problem Set 0 that walked students through enough Python syntax to make a few basic plots. The idea was to show that a handful of commands could generate a figure, as an alternative to a lot of pointing and clicking in Excel. (If you are curious, that old problem set is still on the websites for previous years.) But, of course, the world has changed.

We are all a little more like Prof. Greg Voth now, who once remarked that he programs in the very high-level language called “graduate students.” Claude, Codex, and their relatives have become *extremely* good at taking natural-language instructions, reducing them to code in a high-level language like Python, and reducing that in turn to operations on actual ones and zeros. It is a fun and empowering development, and it is also a little sad for you. In the same way that I had to trust that my compiler faithfully turned my C++ into a binary, you now have to trust that your English cascaded all the way down to the bits correctly. There is a difference, though. Compilers almost never get it wrong. Chatbots do, and often. The remedy is to move your trust down a level, to code you can read, so that the only thing you take on faith is Python itself. So this year’s Problem Set 0 is shorter than it used to be, and it is about that skill. The hard part is no longer getting the computer to do something. The hard part is knowing whether it did what you think it did.

You may use any tool you like for this and for the computational parts of the later problem sets, including chatbots. (I mean it. I am not going to police it, and I would rather you used the tools well than pretended you didn’t.) The one thing I ask in return is that you understand what you submit. At the end of the quarter we will sit down and talk about your problem sets, and I may ask you to walk me through a piece of code you turned in.

Here is the computational experiment we will use throughout. Flip a coin that lands heads with probability $p = 0.7$ a total of $n = 100$ times and record the fraction of heads. Repeat that whole experiment 5,000 times so that you have 5,000 fractions. Make a histogram of them and report their mean and standard deviation.

1. **Predict, then ask for the plot.** Before you touch a computer, **write down what you expect for the mean of the 5,000 fractions and for their standard deviation.** (The second one takes a little thought. The variance of a single flip is $p(1 - p)$.) Now describe the experiment to a chatbot (Claude, ChatGPT, Gemini, whichever you like) and ask it for the histogram, the mean, and the standard deviation. Most of them will quietly write and run code in a sandbox of their own and hand you back a picture and two numbers. **Did the numbers agree with your prediction?** Notice how much you had to take on faith to get here. You do not know what the chatbot ran, whether the coin was really biased at 0.7, whether it flipped 100 times or 5,000, or whether the “standard deviation” it reported is the one you asked for.
2. **Ask for the code.** Now ask the chatbot for the Python code that produced the plot. Then push back on it. Insist on short, plain code, with no clever one-liners and no library call whose behavior you cannot state in a sentence. If there is a line you cannot restate in your own words, make the chatbot rewrite it more plainly until you can.
3. **Read the code.** Your goal is to be able to read the code, line by line, and confirm from the code alone that it does the experiment described above, so that the only thing you are trusting is that Python does what Python says, and not that the chatbot did what the chatbot says. **Write a comment on every line, in your own words, saying what that line does.** (Do not let the chatbot write the comments. That defeats the purpose.)
4. **Run it yourself.** Get some environment running in which you can execute that code and see the plot appear. The easy default is Google Colab (colab.research.google.com), which needs no installation and is the one environment Camelia and I will actively help you with. A Python installation on your own laptop is fine too, and a chatbot will happily walk you through setting one up, but if it breaks at 11 pm you are the one maintaining it. (Later problem sets hand out starter code as Jupyter notebooks, which open in Colab with one click and elsewhere with a little more effort.) You could also keep running everything inside the chatbot’s sandbox, and for some of the later problem sets that will be perfectly adequate, but then the sandbox is one more thing you are trusting, and you cannot poke at it. The point is that when Problem Set 1 asks you to plot something, I want you to know that there are options other than (better than?) running to Excel. Run the code, **confirm that the mean and standard deviation match what the chatbot reported in part 1, and save the histogram twice, once as a PNG and once as a PDF, then get both files onto your own computer.** Zoom way in on each image. The PNG is a raster image, a grid of colored pixels, so past some magnification the lines turn to mush. The PDF is vector graphics, a list of instructions for drawing lines and curves, so it stays sharp at any zoom. Vector formats (PDF, SVG, EPS) are what you want for a plot in a writeup or a paper; raster formats (PNG, JPEG) are for photographs, screenshots, and scatter plots with so many points that a vector file would be enormous.
5. **Change one thing.** Change n from 100 to 400 in the code you now understand. **Before you rerun, say how the standard deviation should change and why.** Then rerun and check.

That’s it. If you want feedback, upload the notebook or a PDF to Canvas and Camelia or I will take a look. If the setup itself defeats you, bring your laptop to office hours, which is exactly what they are for.