Annotations will be typeset in red. Red footnotes are also annotations and not part of the syllabus proper.

Key information

Instructor	
Name:	Josh Laughner
Pronouns:	he/him/his
Email:	jlaugh@caltech.edu
Office:	126 Linde+Robinson
Office hours:	Mon 3p-4p Tue 5p-7p Thu 10a-11a By appointment

Class	
Lecture:	130 Linde+Robinson 10a-11a MWF
Website:	`courses.caltech.edu/ds01`
Textbook:	None. Readings and other resources will be posted to the course website.

I have not had luck yet finding a CS textbook that is well-tailored to scientific programming. I'm looking into data science books, but for now plan to assign individual resources as needed.

Course description

DS 01 is designed to teach you the fundamentals of computer programming in Python, with an emphasis on using it for data analysis. No prior programming experience is required or expected. We will start with the absolute basics, then cover some tools available to work with more complex datasets, and end with how to balance writing maintainable and reusable code against the rapid evolution of what your code needs to do while in the midst of an analysis project. DS 01 is a project driven course.

Learning objectives

By the end of DS01, you will be able to:

1. Understand the purpose of literals, variables, loops, functions, modules, and packages
2. Use Python to identify underlying patterns in datasets of any size
3. Create a reproducible scientific analysis
4. Include good programming practices in their data analysis
5. Identify when it is appropriate to use rough "single-use" code and when it is better to take time to refine their code.

From the standpoint of backwards design, it's clear that my highest priority for the students is for them to learn how to write good code, since #4, #5, and arguably #3 all relate to that. It may seem strange then that this idea of good code doesn't really show up explicitly until the last unit in the schedule at the end. The challenge is I doubt students will really internalize the importance of writing good code until they experience the cost of writing bad code. Plus, I'm not assuming no prior programming experience, and simultaneously learning syntax and how to conceptualize a solution to a problem as an algorithm are strenuous enough without adding standards of coding on top of that.

To compromise, I've tried to organize the activities and assessments so that some of the ideas of good programming practice are present throughout without being explicit. For example, the first unit project is a library of functions, encouraging code reuse, and when we introduce functions, we'll go over what role they play in coding. The final project will also incorporate a lot of emphasis on reusing code from earlier in the semester to give students the experience of trying to reuse probably bad code. The idea is to give them that visceral sense of why this matters.

Course welcome

Welcome to DS01! Our goal this semester is to help you learn how to use compute programming to carry out data analysis in your favorite STEM field. Data analysis is a critical skill for any STEM discipline, as more and more graduates are expected to be able to sort through large datasets efficiently and discern the key patterns or underlying mechanisms behind the data.

This class is divided into three units. In the first unit, we will focus on the fundamentals of programming. This includes both the technical elements (the syntax that the computer requires you follow) and more conceptual ideas (how to translate a process into an algorithm). The second unit will introduce tools geared towards data analysis, such as plotting, fitting, reading/writing data files, and using more powerful data structures such as dataframes and arrays. The final unit will step back from the details of programming and data analysis and look at good programming practices to follow, when it's important to write good code vs. get something done quickly and messily, and how this ties into expectations of scientific journals to make your code available for peer review.

Help with resources

As a programming class, you will need access to a computer to complete the assignments. A laptop that you can bring to class is ideal for the in-class activities, although there will be ways for you to participate without one. If you do not have reliable access to a computer, please let me know so we can make arrangements for you.

When I was in undergrad, there were several buildings on campus that had open computer labs for students. I hope this will still be an option in the future, as well as support to have laptops available to loan to students who need them.

Getting help with the class

I'm assuming that the students in the course will be in their first two years, so some of them may not be familiar with how office hours work.

At times throughout the semester, you may feel frustrated, overwhelmed, or just stuck. *That is normal.* Your TAs and myself are here to help you. There's a number of ways that you can get help from us:

Office hours: are a time when you can come talk to us in person, no appointment needed. Just come by during one of the times listed as office hours and we'll be happy to help. My office and office hours are listed at the top of the syllabus; for the TA office hours, see the course website. If none of the listed times work for you, feel free to contact us to set up a time to meet.

Email: You can also email myself or one of the TAs for help directly. We'll do our best to get back to you within a working day. You can help us out by being clear and concise about what help you need. If you're confused about something from lecture for example, telling us what you did understand, what you didn't understand, and pointing to the relevant slide numbers in a few sentences will help us help you faster.

Online resources: There are many online resources where you can find help as well. `stackoverflow.com` is a site where anyone can ask a (usually computer-related) question and get answers back from the community. `geeksforgeeks.org` also sometimes has nice examples for individual functions available within different languages. Usually searching for "python" plus an error message or a few words describing what you want to do in your favorite search engine will return hits from these sites. There are of course many other sites, but these are ones I visit myself. When you find help on these site though, make sure you understand why the piece of code they suggest works, don't just copy-paste.

Each other: I strongly encourage you to seek out help from each other as well as the TAs. Often someone who has just learned something can describe it in a more helpful way than someone who has known it for a long time (look up "expert amnesia" if you're curious). Also, teaching someone else something is one of the best ways to cement that concept in your own mind. By all means review the material or work together on homework with your friends (but see the collaboration policy below).

Collaboration policy and honor code

While I want you to all feel comfortable working together and helping each other, I also expect everyone to complete their own assignments. The assignments are designed to reinforce and give practice with the concepts we cover in class, and if you don't do them yourself, you won't get the full benefit of the class.

For assignments and projects, I expect each of you to turn in your own, original work. That means:

- *Do* discuss the course material (lecture, readings, or supplemental) with each other.
- *Do* brainstorm together about how to approach a problem on the homework.
- *Do* help each other debug code (but the author should try themselves first).
- *Do not* send code to each other.
- *Do not* copy-paste code that someone else sends you.
- *Do not* copy-paste solutions to the problems that you might find online.

Caltech has a campus-wide academic integrity policy and honor code, as follows:

- No member of the Caltech community shall take unfair advantage of any other member of the Caltech community.
- Plagiarism is the appropriation of another person's ideas, processes, results, or words without giving appropriate credit, and it violates the honor code in a fundamental way. You can find more information at: http://writing.caltech.edu/resources/plagiarism.
- All instances of plagiarism or other academic misconduct will be referred to the Board of Control for review.

Feeling sick?

If you're ill, please stay home and get better. All lecture slides will be posted, and I or the TAs will be happy to meet with you to review those slides once you're recovered. If you need to miss a day with in-class activities (see the subsection under Assignments), email me to let me know you won't be in class and we will arrange a make up activity.

Assignments

Assignment	Percent of final grade
Homework (all)	50%
In-class activities (all)	10%
Unit 1 project	10%
Unit 2 project	10%
Unit 3 project	20%

Homework

There will be 10 homework assignments over the course of the semester. Late work will receive $\frac{1}{4}$ credit. However, as long as you turn in all 10 assignments **by the last Friday of classes** your lowest two homework grades will be dropped and the remaining 8 will equally contribute to your homework grade. I may grant extensions if there are extraordinary circumstances (such as a family or medical emergency), contact me directly to discuss this.

The homework in this class is designed to reinforce the skills you learn in class and to build competence with each skill needed in order so that you're ready for the next task. That's why it's important to complete the assignments on time, so that you have a firm foundation for the following topics.

In-class activities

Some classes will have practical components, supervised in-class work. Being in class and participating in these activities will count towards the 10% of your grade for in-class activities. These class periods are marked with a "P" type on the schedule below. Similar to homework, you may miss **one** with no loss of points and can make up **two** on your own outside of class, no questions asked, **if you contact me before the class**. If you need to make up more than that outside of class, please schedule time with me so we can make sure your learning goals for the course will be met.

Projects

There will be three projects throughout the semester. These will be larger programs or data analyses that will require you to combine everything that you have learned up to that point in the class.

- *Unit 1 project:* create a library of useful functions as a package. You will be given a

list a 10 desired functionalities (for example, compute a relative difference) and need to write a function to handle each one.

- *Unit 2 project:* this project will focus on analyzing a large datasets to identify trends or anomalies or to compute useful statistics. There will be several options available in different fields, and you will be able to choose the one most interesting to you.
- *Unit 3 project:* will be an extension of the second project and will expect you to incorporate some of the library you wrote for the first. This will require you to dig deeper into some of the correlations or causal relationships between variables in your chosen dataset.

The **Unit 1 project** is intended to get the students to write a reasonably large chunk of code, but broken up into parts that are not interdependent, hence the library of functions. The functions are also going to be small enough that (for the most part) they don't need broken up into multiple functions, for example:

- `reldiff`: compute a difference $b - a$ relative to a or the mean of a and b.
- `absmax`, `absmin` given a sequence of values, return the value farthest (closest) to 0.
- `bin_1d`: bin y by x and return the result of a desired operation (count, mean, max, min, etc.) on each bin.
- `wrap_string`: wrap a string so that individual lines do not exceed n characters, splitting on whitespace

The other goal behind this project is to get them to create functions that they'll be able to reuse for the later projects, possibly even the homework. This is to introduce them to the mind set of how reusing code can really save time. This will be the main assessment for learning goal #1, since it will require them to put together all the programming basics to create.

The **Unit 2 project** is geared towards getting the students comfortable working with various types of data files, e.g. text, netCDF, or field-specific formats, as well as plotting and fitting data, and understanding what the result tells them. I'd have probably 3 topics to choose from, ideally selected so that the majority of students in the class would have one in or adjacent to their field. Examples from atmospheric science could include calculating the Niño index or the NO_2 anomaly over China this winter (it's way, way lower than usual). This will be the main assessment for learning goal #2.

The **Unit 3 project** will, ideally, extend the Unit 2 project. The goal would be to get the students to reuse their code from both the previous two projects. For example, if they calculated the Niño index in project 2, they could look at how that correlates with rainfall/temperature in the Americas. The second part of the assignment will be a short ($\sim$ 1 page) write up from each student describing when they followed good programming practice and how and when they didn't and why. I'll also have them submit this as closely

as possible as they would if it were supporting a journal article—including getting a DOI if I can find a way to make a temporary/dummy DOI.

I'm also trying to set this up to make students experience the cost of poorly written and documented code. Because they have to go back and document their Unit 2 project for the last homework, but *not* their Unit 1 project, but use both sets of code in the Unit 3 project, my hope is that they'll have the visceral experience of "how the heck does *this* function work!?" for their Unit 1 code when they have to use it in Unit 3. That will definitely be something I need to evaluate in the student feedback at the end of term, either as part of course feedback or part of the writeup they submit with the project.

This will be an assessment of learning goals #3–5.

Course schedule

This section assumes semesters, rather than quarters or terms (even though Caltech is on terms).

One pedagogical point that stuck with me is that a new professor will have very limited time for course development, so I've only included a limited number of classes mostly or entirely centered on active learning, since these take more time to develop well. Most of these involve pair or group work to solve a collection of problems and then reporting out what they did to solve the problem and why to the rest of the class. Having clear roles (e.g. one person edits the code, one person records the thought process and reports, the others focus on the programming) will definitely help. It would also be nice for students to rotate roles, so that different students get to be the programmer/reporter/etc., but I think the activities are spaced too far apart for them to remember what role they did last in the present form.

Ideally, there would be many more activities so student would be better able to rotate roles. This would also be benenficial since the only way to learn programming is to do it. However, this is designed for a pre-tenure position that doesn't have time to build a fully flipped class yet. I would adapt it over time as I learn about the student culture at my institution.

Class type: the "type" column indicates whether a class is lecture (L) or practical (P). Practical classes will consist of active in class activities which will contribute 10% of your grade. **Bring a laptop to practical classes.**

		Type	Topic	Assignment due
Unit 1: programming basics				
Week 1	Mon	L/P	Course introduction, Python installation	
	Wed	L	Jupyter notebooks vs. scripts	

		Type	Topic	Assignment due
	Fri	L	Literals, variables, variable types, operators, and comments	
Week 2	Mon		*Holiday*	HW1 (identifying code basic components)
	Wed	L	Code blocks (`if`, `for`, `while`)	
	Fri	L	Importing packages; finding and understanding documentation	HW2 (using code blocks)
Week 3	Mon	L	Thinking like a computer 1	
	Wed	P	Thinking like a computer 2	
	Fri	L	Practical example: simple physics model	HW3 (packages)
Week 4	Mon	P	Practical example: simple physics model	
	Wed	L	Functions: definition, inputs, outputs, and variable scope	
	Fri	L/P	Functions: when to use	HW4 (algorithm/pseudocode)
Week 5	Mon	L	Code organization: modules and packages	
	Wed	L	Debugging 1	
	Fri	P	Debugging 2	HW5 (physics model)
Week 6	Mon		Catch up/extra topics	
	Wed		Catch up/extra topics	
			Unit 2: data techniques	
	Fri		Arrays (creation and indexing)	
Week 7	Mon	L	Pandas dataframes	
	Wed	L	Reading/writing text files	
	Fri	L	Reading/writing binary files and pickling	Unit 1 project
Week 8	Mon	L	Reading scientific data: HDF5 and netCDF files	
	Wed	L	Plotting 1 (figures; axes; line and scatter plots)	
	Fri	L	Plotting 2 (pcolor, contour, histograms, multipanel plots)	HW6 (data)
Week 9	Mon	L	Fitting data 1 (linear regression, regression types)	
	Wed	L	Fitting data 2 (arbitrary functions)	
	Fri	L	Optimization/minimization	
Week 10	Mon	L	Advanced dataframes 1: fancy indexing, interpolation, and times	HW7 (plotting and fitting data)
	Wed	L	Advanced dataframes 2: joins and groupbys	
	Fri	L	When to use what data structure	HW8 (optimization)
Week 11	Mon	L	Catch up/extra topics	
	Wed	L	Catch up/extra topics	
			Unit 3: good programming practices	
	Fri	L	Documenting your code with docstrings	HW9 (dataframes)

		Type	Topic	Assignment due
Week 12	Mon	L	Self documenting (readable) code	
	Wed	L	Breaking up code for readability	
	Fri	L	Enabling code reuse	Unit 2 project
Week 13	Mon	L	Automatic testing 1	
	Wed	P	Automatic testing 2	
	Fri	L/P	Version control 1[1]	
Week 14	Mon	P	Version control 2	
	Wed	L	Isolating projects in environments	
	Fri	L	Publishing research code	HW10 (project 2 docs and unit tests)
Week 15	Mon	L	Fast or right?	
	Wed	L	Catch up/extra topics	
	Fri	L	Catch up/extra topics	
Finals	Fri			Unit 3 project

Expanding on class topics & ideas for practical classes:

- *Course intro/installation:* overview of course goals and structure plus walking through installing Python and Jupyter Notebooks on students' computers.
- *Notebooks vs scripts:* describe the features of notebooks, contrast with standard `*.py` script files.
- *Literals, variables, etc.:* cover the basic elements that make up any programming language.
- *Code blocks:* cover syntax and uses of `for` and `while` loops and `if-elif-else` blocks.
- *Importing packages:* how to import additional packages to use. How to understand their documentation.
- *Thinking like a computer 1 & 2:* trying to separate syntax from algorithm design by going through how a problem must be described to a computer in pseudo-code. The second period will be small group activities where each group is given a STEM-related problem (e.g. matrix determinant, computing degree of unsaturation in a molecule, etc.) that they write pseudo-code for and then explain their algorithm to the class.
- *Simple physics model:* join algorithm design with real syntax, explain how simple kinetics equations can be modeled numerically to solve where a cannonball will land.

[1] After discussion with a CS lecturer, whose opinion was that introducing too many programming tools in the first course students take is counterproductive, I decided I would rather get the students a firm foundation in programming and plant a seed for version control rather than trying to stress this throught the whole class.

The second period will have students work in small groups to implement this and check it against the known solution.

- *Function definitions etc:* the basic syntax of defining a function, how data are passed into and out of functions, and the rules of variable scope.
- *When to use functions:* discussion of when to use functions/the role they play in programming.
- *Code organization modules and packages:* How to create your own modules and packages that can be imported into a notebook.
- *Debugging 1 & 2:* talking through a process for debugging crashes or bugs. Second period will involve small group work (or pairs) to debug a simple problem then have one or two groups talk through their process with the class.
- *Arrays:* creating and indexing Numpy multi-dimensional arrays
- *Pandas dataframes:* creating and indexing Pandas dataframes (similar to R dataframes), pros and cons relative to arrays
- *Reading/writing text files:* how to open, read, and parse text files. Will cover both manually parsing and using Pandas tools for .csv files
- *Reading/writing binary files:* how to open, read, and parse binary files. Cover "pickling" which is a way of writing arbitrary Python variables to disk.
- *Reading scientific data:* how to read HDF5 and netCDF4 files, which are hierarchical data formats especially common in earth sciences. Discussion of importance of open source, common data formats.
- *Plotting 1 & 2:* covering how to make plots in Python. First period will cover the two ways (function vs. object-oriented) of plotting in Python and basic line/scatter plots). Second period will get into additional plot types and basics of combining multiple axes into a single figure.
- *Fitting data 1 & 2* explain how to do standard y-residual fits, RMA fits, and robust fits; discuss the difference between them. Second period discuss how to use the `scipy.optimize.curve_fit` function to fit data to an arbitrary function.
- *Optimization/minimization* discuss the concept of cost functions and how to minimize one, give example applications.
- *Adv. dataframs 1 & 2:* multilevel indexes, interpolating data, datetime indexes in the first period. Second period will focus on SQL-like joins and how to quickly do group and reduce operations, e.g. monthly averages.
- *When to use what data structure:* discussion of when to use lists, dicts, tuple, set, arrays, dataframes, etc. as well as the importance of being consistent throughout a project to avoid messily converting between types as much as possible.

- *Documenting with docstrings:* discussion of best practices for documenting functions and modules.
- *Self documenting code:* best practices for self-documenting code; descriptive variable and function names, whitespace, making intent clear
- *Breaking up code for readability:* go through rules of thumb for when a single function should be broken up into multiple functions to improve maintainability and readability
- *Enabling code reuse:* how to distill problem-specific code into more generally reusable code; general core function with problem-specific wrappers; how to recognize when to take the time to write a general function.
- *Automatic testing 1 & 2:* how to write automatic tests to ensure that a function does what it is meant to. Distinction between unit and integration testing, with emphasis on the challenges for scientific code (i.e. change is expected in some cases—how to ensure the correct change occurred). Second period workshop style class where students have opportunity to write tests for their Unit 1 project functions, getting help from each other, TAs, and me.
- *Version control 1 & 2:* introduction to what version control is and why it matters. Basic tutorial on tracking changes in the code over time.
- *Isolating projects in environments:* concept of a Python environment and how to integrate with notebooks, advantages of isolating research projects in individual environments.
- *Publishing research code:* how to organize your research project code so that is it easy to upload in support of a paper; expectations of journals and the scientific community.
- *Fast or right:* discussion of when it's important to take the time to build clean, maintainable code, and when it's better to hack together something that works once for a particular analysis.
- *Catch up/extra topics:* I'm virtually certain I'll get behind at some point in the semester and I want these buffers to not feel rushed. If don't need the time to catch up, then can use these for extra topics (e.g. `xarray` for multidimension arrays with labeled dimensions/coordinates in Unit 2) or class workshop time for students to get help on projects.

Homeworks:

- Homework 1: given individual lines of code, ask the students to identify what each term in the line is.
- Homework 2: two parts. First, given a scenario, ask students to identify which of a `for` loop, `while` loop, of `if` statement is appropriate. Second, ask them to write the start of these blocks for various situations (e.g. loop while `s` is a string)

- Homework 3: have students import packages in various ways (import the package itself, import the package but rename it, import a function from a package), then have them use various simple functions from packages that we have not used yet to practice reading the documentation and applying it.
- Homework 4: have students write out pseudocode (that explains the steps in the algorithm but is not in strict Python syntax) for 3 problems, e.g. testing the Collatz conjecture for any positive integer, multiplying two matrices, etc.
- Homework 5: have students finish a numerical model of a cannonball fired with some initial velocity and position to compute where it lands. We will create the basics in class, they will have to extend the model (e.g. implementing curved rather than flat ground) themselves.
- Homework 6: given a text file, binary file, and netCDF file, students will need to read in some of the data in the file and do some basic calculation on it (mean, standard deviation, etc).
- Homework 7: using data files of some sort, students will need to make line, scatter, hist, pcolor plots of the data. They will also need to fit linear regression to various parts of the data and interpret the slope/intercept.
- Homework 8: Given a dataset, the students will need to create a cost function to minimize to find the best combination of parameters. This could be trying to minimize two or three types of non-orthogonal error, or choosing an apartment that is the ideal balance between rent, commute, and living space.
- Homework 9: Given multiple datasets, the students will need to use the database-like join and grouping operations available to dataframes to combine these datasets to answer a series of questions. For example, given one table of apartment rents, one of square footage, and one of median rents per square foot by zip code, the students will need to match individual rents to median rents and analyze the distribution within each zip code (e.g. is it a fairly normal distribution or long-tailed?)
- Homework 10: the students will need to take their Unit 2 project code, add documentation, and add unit tests to ensure small functions do what they are supposed to.